```
template<class Sq>
class Game //classe abstraite
```

```
public:
Game(int,int); //dimensions
virtual void play();
virtual void demo();
virtual ~Game();

protected:
const int height;
const int width;
vector<Sq>* plateau;
long long score;
```

```
private:
bool quit;
virtual void init()=0;
virtual bool is_over() const=0;
virtual void move(Direction)=0;
virtual void print_board(ostream& o=cout) const;
friend ostream& operator<<(ostream& o, const Game<Sq>& game);
virtual void move_up();
virtual void move_down();
virtual void move_left();
virtual void move_right();
virtual bool is_stuck() const;
```

```
enum class Direction { up, down, left, right}
```

class Game_2048 : public Game<Square_2048>	template <class C> class Taquin : public Game<Square_Taquin<C>>	class Sokoban : public Game<Square_Sokoban>
public:	public:	public:
Game_2048(int height);	static const Square_Taquin<C> empty;	**Sokoban**(int h,int w, int nb_crates=-1);
	Taquin(int,int);	virtual **~Sokoban**();
protected:	virtual **~Taquin**();	
virtual Square_2048 **random_square**() const;		protected:
virtual unsigned long long **random_value**() const;	private:	static const int min_height=10;
	int pos_empty_w;	static const int min_width=10;
private:	int pos_empty_h;	int nb_crates;
bool board_change;	virtual void **init**();	int pos_h;
vector<pair<int, int>> empty_squares;	virtual bool **is_over**() const;	int pos_w;
virtual void **init**();	virtual void **move**();	int i_top_left;
virtual void **move**(Direction dir);	void **fill**();	int j_top_left;
virtual bool **is_over**() const;	void **mix**();	int i_top_right;
void **transpose_board**();		int j_top_right;
void **pop_up_new_square**();		int i_bottom_left;
void **slide_line**(int i, Direction dir);		int j_bottom_left;
void **merge_line**(int i, Direction dir);		int i_bottom_right;
void **add_empty_square**(int i, int j);		int j_bottom_right;
template<class It>		virtual void **print_board**(ostream& o=cout) const;
int **slide_line_template**(It begin, It end);		virtual void **init**();
void **slide_board**(Direction dir, bool transpose);		virtual void **set_walls**();
template<class It>		virtual void **setExternalWalls**();
void **merge_line_template**(It begin, It end);		virtual void **setInternalWalls**();
template<class It>		virtual void **set_target_crates**();
int **slide_merged_line**(It begin, It end);		virtual bool **free_zone**(int h_c, int l_c) const;
virtual bool **is_mergeable**(Square_2048& sq) const;		virtual bool **outsideOfWalls**(int h_c, int l_c) const;
virtual Square_2048 **merge**(Square_2048& sq);		virtual void **move**(Direction s);
		virtual void **set_pers**();
		virtual bool **is_over**() const;
		virtual bool **is_stuck**() const;

class Game_2048_Num : public virtual Game_2048	class Game_2048_Neg : public virtual Game_2048

public:
Game_2048_Num(int height, int base=2);
 	
protected:
const int base;
virtual unsigned long long **random_value**() const;
	public:
	Game_2048_Neg(int height);
	
	protected:
	virtual Square_2048 **random_square**() const;

class Game_2048_Mult : public virtual Game_2048	class Game_2048_Dest : public virtual Game_2048

public:
Game_2048_Mult(int height);
 	
protected:
virtual Square_2048 **random_square**() const;
	public:
	Game_2048_Dest(int height);
	
	protected:
	virtual Square_2048 **random_square**() const;

class Game_2048_Num2 : public virtual Game_2048	class Game_2048_Mix : public Game_2048_Num2, public Game_2048_Neg, public Game_2048_Mult, public Game_2048_Dest
public: **Game_2048_Num2**(int height); protected: const int base; virtual unsigned long long **random_value**() const;	public: **Game_2048_Mix**(int height, int base); protected: virtual Square_2048 **random_square**() const;

```
class Printable //classe abstraite

public:
friend ostream& operator<<(ostream& out, const Printable& object);

private:
virtual void print(ostream& out) const = 0 ;
```

```
class Square_2048 : public Printable

public:
static Square_2048 empty;
Square_2048(Square_2048_action action = empty, unsigned long long value
=0);
bool operator==(const Square_2048& sq) const;
bool operator!=(const Square_2048& sq) const;
bool dest_possible(const Square_2048& sq) const;
bool mult_possible(const Square_2048& sq) const;
bool is_opposite(const Square_2048& sq) const;
bool same_action(const Square_2048& sq) const;
bool same_value(const Square_2048& sq) const;
Square_2048& operator=(const Square_2048& sq) const;
void set_value(unsigned long long value);
unsigned long long get_value() const;
void swap(Square_2048& sq);
bool is_empty() const;

private:
Action_2048 action;
unsigned long long value;
virtual void print(ostream& out) const;
```

```
template<class C>
class Square_Taquin : public Printable

public:
static const Square_Taquin<C> empty;
Square_Taquin(unsigned long l=0);
Square_Taquin(const Square_Taquin<C>& sq);
bool operator==(const Square_Taquin<C>& sq) const;
bool operator!=(const Square_Taquin<C>& sq) const;
bool operator<(const Square_Taquin<C>& sq) const;
bool operator<=(const Square_Taquin<C>& sq) const;
bool operator>(const Square_Taquin<C>& sq) const;
bool operator>=(const Square_Taquin<C>& sq) const;
Square_Taquin& operator=(Square_Taquin<C>& sq);
Square_Taquin& operator++();
Square_Taquin& operator++(int);
Square_Taquin& operator--();
Square_Taquin& operator--(int);

private:
virtual void print(ostream& o) const;
unsigned long value;
```

enum class Action_2048 { empty, none, neg, mult, div, destroy }
string **to_string**(Action_2048 action);

enum class Square_Sokoban { empty, wall, pers, crate, target, crate_target, pers_target }
ostream& **operator<<**(ostream& out, Square_Sokoban const& c);