Player.h

```
struct Position {
	int row;
	int col;

	bool operator==(const Position &other) {
		return row == other.row && col == other.col;
	}
};


class Player {
public:

	Player(const std::string name, const bool is_human);

	std::string get_name() const {return name_; }
	int get_points() const {return points_; }
	Position get_position() const {return pos_; }
	bool is_human() const {return is_human_; }

	void ChangePoints(const int x);

	void SetPosition(Position pos);

	std::string ToRelativePosition(Position other);

	std::string Stringify();

private:
	std::string name_;
	int points_;
	Position pos_;
	bool is_human_;

}; // class Player
```

Maze.h

```
enum class SquareType { Wall, Exit, Empty, Human, Enemy, Treasure
};

std::string SquareTypeStringify(SquareType sq);

class Board {
public:
	Board();

	int get_rows() const {return 4; }
	int get_cols() const {return 4; }

	SquareType get_square_value(Position pos) const;

	void SetSquareValue(Position pos, SquareType value);

	std::vector<Position> GetMoves(Player *p);

	bool MovePlayer(Player *p, Position pos);

	SquareType GetExitOccupant();


	friend std::ostream& operator<<(std::ostream& os, const Board &b);

private:
	SquareType arr_[4][4];

	int rows_;
	int cols_;

};  // class Board

class Maze {
public:
	Maze(); // constructor

	void NewGame(Player *human, const int enemies);

	void TakeTurn(Player *p);

	Player * GetNextPlayer();

	bool IsGameOver();

	std::string GenerateReport();
	friend std::ostream& operator<<(std::ostream& os, const Maze &m);

private:
	Board *board_;
	std::vector<Player *> players_;
	int turn_count_;

};  // class Maze
```