```
#ifndef _GAME_H_
#define _GAME_H_

#include <vector>
#include "Player.h"

// you may change this enum as you need
enum class SquareType { Wall, Dots, Pacman, Treasure, Enemies, Empty, PowerfulPacman
, Trap, EnemySpecialTreasure };

// TODO: implement
std::string SquareTypeStringify(SquareType sq);

class Board {
public:
        // // TODO: implement
        // Board();
        // this would be a default 10 x 10 board



        // Board(int rows, int cols);
        // this can be a constructor that sets the board size




        // // already implemented in line
        // int get_rows() const {return 10; }  // you should be able to change the s
ize of your
        // default board by changing these numbers and the numbers in the arr_ field
        // int get_cols() const {return 10; }  // board by changing these numbers an
d the numbers in the arr_ field


        // // TODO: you MUST implement the following functions
        // SquareType get_square_value(Position pos) const;

        // // set the value of a square to the given SquareType
        // void SetSquareValue(Position pos, SquareType value);
        // look in the array and set the value of the square to the given SquareType
        // this will use the row n

        // // get the possible Positions that a Player/Enemy could move to
        // // (not off the board or into a wall)
        // std::vector<Position> GetMoves(Player *p);
        // this will return a vector of possible positions that a player can move to
        // this will be based on the current position of the player and the board
        // this will also take into account the walls and the other players
        // I rewrote this to just use rows and columns
        // i would return this below:
        // std::vector<std::pair<int, int>> positions;
        // the function would be:
        // std::vector<std::pair<int, int>> GetMoves(Player *p);
        // this will return a vector of possible positions that a player can move to
 in the coventional x and y coordinates as ints







        // // Move a player to a new position on the board. Return
        // // true if they moved successfully, false otherwise.
        // bool MovePlayer(Player *p, Position pos, std::vector<Player*> enemylist);
        // this will move the player to a new position on the board
        // this will return true if they moved successfully, false otherwise
        // this will also take into account the other players and the walls
        // this will also take into account the enemies
        // this will also take into account the dots and the treasure
        // if i rewrite this i would have:
        // bool MovePlayer(Player *p, std::pair<int, int> pos, std::vector<Player*>
enemylist);
        // this would just use the conventional int pair cordinates






    // // Move an enemy to a new position on the board. Return
        // // true if they moved successfully, false otherwise.
    // bool MoveEnemy(Player *p, Position pos);
        // this will move the enemy to a new position on the board
        // this will return true if they moved successfully, false otherwise
        // this will also take into account the other players and the walls
        // this will also take into account the enemies
        // this will also take into account the dots and the treasure

        // if i rewrite this i would have:
        // bool MoveEnemy(Player *p, std::pair<int, int> pos);





        // // You probably want to implement this
        // friend std::ostream& operator<<(std::ostream& os, const Board &b);
        // this will print the board to the console
        // this will be based on the current state of the board
        // this will also take into account the players and the enemies
        // this will also take into account the walls and the dots
        // this will also take into account the treasure and the traps
        // this will also take into account the powerful pacman and the enemy specia
l treasure


private:
        SquareType arr_[10][10];//this is how the board is stored, this is based on
the default constructor
        // if you want to set to another size then this will be set based on the inp
ut
        // SquareType arr_[rows][cols];

        int rows; // might be convenient but not necessary
        int cols;
        // this is the amount of rows and columns in the board

        // you may add more fields, as needed
};  // class Board

class Game {
public:
        // TODO: implement these functions
        // Game(); // constructor
        // seems like this would not get used often so i would probably not implemen
t this




        // // initialize a new game, given one human player and
        // // a number of enemies to generate
        // void NewGame(Player *human,std::vector<Player*> enemylist, const int enem
ies);
        // this will initialize a new game
        // this will take in a human player and a list of enemies
        // i think that it might be better to not pass in the list of enemies becaus
e they wouldn't be generated yet
        // this will also take in the number of enemies to generate
        // and return the list of hte enemies generated


        // // have the given Player take their turn
        // void TakeTurn(Player *p,std::vector<Player*> enemylist);
        // this will have the given player take their turn
        // this will also take in the list of enemies
        // i think that this should not need to take in the list of enemies because
the enemies should be on the board
        // since the board should know
        // this will also take into account the other players and the walls

        // //have the enemy take turn
        // void TakeTurnEnemy(Player *p);
        // this will have the enemy take their turn
        // bool IsGameOver(Player *p);
        // this will check if the game is over







        // // return true if all pellets have been collected
        // bool CheckifdotsOver();
        // this will return true if all the dots have been collected
        // return false if there are still dots on the board



        // // You probably want to implement these functions as well
        // // string info about the game's conditions after it is over
        // std::string GenerateReport(Player *p);
        // this will generate a report about the game's conditions after it is over
        // this will take in the player and return a string
        // this will also take into account the other players and what has all happe
ned during the game



        // friend std::ostream& operator<<(std::ostream& os, const Game &m);
        // this will print the game to the console
        // this will be based on the current state of the game
        // this will also take into account the players and the enemies



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

        // you may add more fields, as needed

};  // class Game

#endif  // _GAME_H_
```