Pacman

I have developed the Pacman Game in Visual Studio Code in the Desktop from udacity. I have not installed further packages, so the dependencies are the same than for the starter code for the Snake Game.

Dependencies for Running Locally (like for the starter code Snake Game)

* cmake >= 3.7
* All OSes: [click here for installation instructions](https://cmake.org/install/)
* make >= 4.1 (Linux, Mac), 3.81 (Windows)
* Linux: make is installed by default on most Linux distros
* Mac: [install Xcode command line tools to get make](https://developer.apple.com/xcode/features/)
* Windows: [Click here for installation instructions](http://gnuwin32.sourceforge.net/packages/make.htm)
* SDL2 >= 2.0
* All installation instructions can be found [here](https://wiki.libsdl.org/Installation)

>Note that for Linux, an `apt` or `apt-get` installation is preferred to building from source.

* gcc/g++ >= 5.4
* Linux: gcc / g++ is installed by default on most Linux distros
* Mac: same deal as make - [install Xcode command line tools](https://developer.apple.com/xcode/features/)
* Windows: recommend using [MinGW](http://www.mingw.org/)

You can build the Pacman Game like follow:

Basic Build Instructions

1. Clone this repo (CppND-Capstone-Pacman).
2. Make a build directory in the folder CppND-Capstone-Pacman: `cd CppND-Capstone-Pacman && mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./PacmanGame`.

Play the game

The user can interact with the game by pressing the following keys on the keyboard:

Key	Action
Up	Move Pacman up
Down	Move Pacman down
Left	Move Pacman to the left
Right	Move Pacman to the right

The Pacman has to eat all pills and must not be caught by the ghosts. Pacman wins if he has eaten all pills.

Project Files

The source files are located in the folder CppND-Capstone-Pacman/src.

character.h / character.cpp
This is a base class from which the classes Ghost and Pacman inherit. It defines methods to extract the start position of the characters from the gameboard file. Furthermore, it stores the current and initial coordinates of the character.

ghost.h / ghost.cpp
This class inherits from the class Character. It provides methods to move the ghost. Furthermore, it contains a member variable which states if the ghost is dangerous or not.

pacman.h / pacman.cpp
This class inherits from the class Character. It contains a member variable which states if Pacman is alive or not.

grid.h / grid.tpp
This class can store a two-dimensional matrix. It provides methods to set and get the values of the matrix.

gameboard.h / gameboard.cpp
This class reads the gameboard file in its constructor. In the gameboard file, there are the positions of the ghosts, of the Pacman, of the food and of the flashing points defined. Furthermore, the gameboard file contains the positions of the walls. In the Gameboard class, grids for the walls, the food, the pills and the flashing points are created.

game.h / game.cpp
This class contains the main game loop. It handles user input from the keyboard to move the Pacman. Next, it contains an update method to update the internal state of the game. During this update method, the positions of the ghosts and of the Pacman are set, it is checked if Pacman has eaten a food, a pill or a flashing point. Furthermore, it is checked if

Pacman was caught or if Pacman caught a ghost. In addition to it, the score is saved. Finally, it calls the render method which takes the game state and renders the state to the screen.

renderer.h / renderer.cpp
This class provides methods to render the state of the game to the screen.

controller.h / controller.cpp
This class provides a method to handle the input from the keyboard. Next, there is a method to move the Pacman given the input direction from the keyboard.

helper.h / helper.cpp
This namespace provides methods to get the directory where the gameboard txt file is saved.

main.cpp
Here, the Renderer, the Controller and the Game are instantiated. The game is started and after playing the game, the result is printed to the console.

Project rubric

Loops, Functions, I/O

Criteria	Meets specification	Addressed in the code
The project demonstrates an understanding of C++ functions and control structures.	A variety of control structures are used in the project. The project code is clearly organized into functions.	See all the classes (Ghost, Pacman, Character, Game, Gameboard, …). Every class contains several methods and member variables. Loops and if-statements are also used. Furthermore, there exists an Enumeration for the directions to move the Pacman and the ghosts.
The project reads data from a file and process the data, or the program writes data to a file.	The project reads data from an external file or writes data to a file as part of the necessary operation of the program.	See the Gameboard class. In gameboard.cpp in line 12, the gameboard txt file is read.
The project accepts user input and processes the input.	The project accepts input from a user as part of the necessary operation of the program.	In the class Controller, the user input from the keyboard is handled. See line 45 in the method HandleInput in the file controller.cpp

Object Oriented Programming

Criteria	Meets specification	Addressed in the code
The project uses Object Oriented Programming techniques.	The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.	See all the classes (Ghost, Pacman, Character, Game, Gameboard, …). Every class contains several methods and member variables.
Classes use appropriate access specifiers for class members.	All class data members are explicitly specified as public, protected, or private.	See all the classes (Ghost, Pacman, Character, Game, Gameboard, …). All class data members are explicitly specified as public, protected, or private.
Classes encapsulate behavior.	Appropriate data and functions are grouped into classes. Member data that is subject to an invariant is hidden from the user. State is accessed via member functions.	See all the classes (Ghost, Pacman, Character, Game, Gameboard, …). Member data is set to private in order to hide it from the user. The member variables are accessed via member functions.
Classes follow an appropriate inheritance hierarchy.	Inheritance hierarchies are logical. Composition is used instead of inheritance when appropriate. Abstract classes are composed of pure virtual functions. Override functions are specified.	See the classes Character, Ghost and Pacman. Ghost and Pacman inherit from the class Character.
Overloaded functions allow the same function to operate on different parameters.	One function is overloaded with different signatures for the same function name.	See the functions CheckCollision in game.cpp.
Templates generalize functions in the project.	One function is declared with a template that allows it to accept a generic parameter.	See the class Grid.

Memory Management

Criteria	Meets specification	Addressed in the code
The project makes use of references in function declarations.	At least two variables are defined as references, or two functions use pass-by-reference in the project code.	See e.g., the function CheckCollision in game.cpp and the function GetPossibleDirections in ghost.cpp.
The project uses smart pointers instead of raw pointers.	The project uses at least one smart pointer: `unique_ptr`, `shared_ptr`, or `weak_ptr`. The project does not use raw pointers.	The GameBoard class defines four shared smart pointers: wall, food, pills and flashingPoints.