Capstone project Snake Game Readme:

1) Project scope:

I decided to improve the snake game based on the snake game example and added following features:

- Implementation of a terminal based user interface to give the user 3 options:
  - play the game
  - see the top 10 high score list
  - quit the game
- Implementation of a high score list, that saves all scores with the user name in a score file, that is located in the same directory as the snake game. For the user names a simple encryption is implemented to avoid manipulation of the score file with a test editor.
  The top 10 high scores can be displayed by the user through the user interface

For this I implemented 3 additional classes:

- Class User, users.h/ users.cpp: the class stores the name and the score of the user and provides a user interface for the input of the username
- Class UserInterface, userinterface.h / userinterface.cpp: the class provides a terminal based interface to give the user the choice what he wants to do: start the game, see the high score list or quit the game
- Class Scores, scores.h / scores.cpp: this class provides the handling of the high score file (read / write including decryption/encryption) and prints out the top ten high scores.

In the main function, the userinterface is started first. Also the path of the SnakeGame is stored in a static variable of the class scores, so the score file will be found, also than the game is started from the finder in macOS.

Depending on the choice of the user the is started (original game code), the high score list will be displayed by calling the print function in the Scores class from the user interface or the game will be terminated.

2) Dependencies for Running Locally

- cmake >= 3.7
  - All OSes: click here for installation instructions

- 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
  - Windows: Click here for installation instructions

- SDL2 >= 2.0
  - All installation instructions can be found here

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

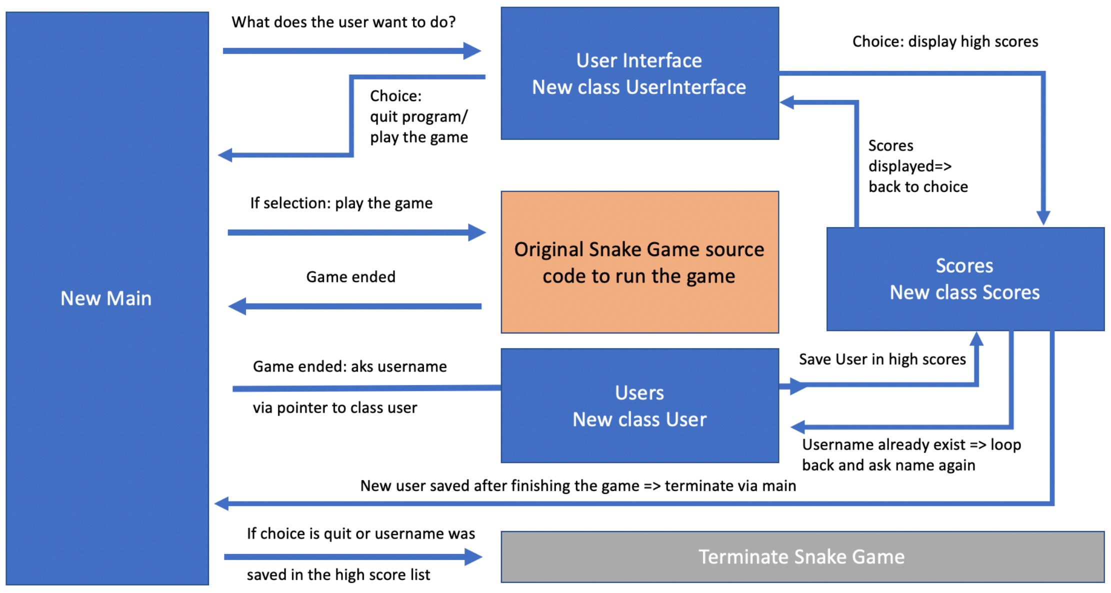
- gcc/g++ >= 5.4

  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - install Xcode command line tools
  - Windows: recommend using MinGW

3) Basic Build Instructions

1. Make a build directory in the top level directory of this project: `mkdir build && cd build`
2. Compile: `cmake .. && make`
3. Run it: `./SnakeGame`.

4) Project scheme:


New Main
What does the user want to do?
User Interface
New class UserInterface
Choice: quit program/ play the game
Choice: display high scores
Scores displayed=> back to choice
If selection: play the game
Original Snake Game source code to run the game
Game ended
Scores
New class Scores
Game ended: aks username
via pointer to class user
Users
New class User
Save User in high scores
Username already exist => loop back and ask name again
New user saved after finishing the game => terminate via main
If choice is quit or username was saved in the high score list
Terminate Snake Game

5) Fulfilled rubric points:

Readme (all rubric points required)

Criteria	Fulfilled	Comment
A README with instructions is included with the project	Yes	
The README indicates which project is chosen	Yes	
The README includes information about each rubric point addressed.	Yes	

Compiling and Testing (All Rubric Points REQUIRED)

Criteria	**Fulfilled**	**Comment**
The submission must compile and run.	Yes	Code was compiled as described in the read me on a local macOS system and runs without errors when started by the terminal or finder

Loops, Functions, I/O

Criteria	**Fulfilled**	**Comment**
The project demonstrates an understanding of C++ functions and control structures.	Yes	The new classes include various control structures like for- or while loops. The new code is clearly organized into functions, e.g. getter/setter in the classes
The project reads data from a file and process the data, or the program writes data to a file.	Yes	In the class Scores (scores.cpp) a read and a write function is implemented. These functions read and save the high score file, which is in the same directory as the SnakeGame. If there is no file (deleted by user or first time the game is started), a new file is created. The file directory is determined by the argv parameter of the main function and saved by a static variable of the Scores class
The project accepts user input and processes the input.	Yes	The class UserInterface implements functions for a user menu, that accepts input and processes the input

Object Oriented Programming

Criteria	**Fulfilled**	**Comment**
The project uses Object Oriented Programming techniques.	Yes	The new project code is organized into 3 new classes (Scores, UserInterface, Users) with class attributes to hold the data, and class methods to perform tasks
Classes use appropriate access specifiers for class members.	Yes	All class data members of the new classes are explicitly specified as public, protected, or private
Class constructors utilize member initialization lists.	Yes	See class UserInterface
Classes encapsulate behavior.	Yes	Appropriate data and functions are grouped into the new 3 classes. Member data that is subject to an invariant is hidden from the user. State is accessed via member functions. E.g. see UserInterface::GetGameStatus and _gameStatusFlag

Memory Management

The project makes use of references in function declarations.	Yes	See Scores::encryptUserName and Scores::decryptUserName
The project uses move semantics to move data, instead of copying it, where possible.	Yes	See main and Scores::addUser
The project uses smart pointers instead of raw pointers.	Yes	See main and Scores::addUser