For my Capstone project I chose to extend the Snake game with additional features. Those features will be described above with more details, but to summarize the basic points of my contribution are:

1. Allow user to choose how many rounds he/she wants to play and if he/she wants to play themselves or have a smart snake instead.
2. Write results and the whole procedure step by step to an external data file in order to allow user see the statistics later on.
3. Create smart snakes, that play the game on their own.

Files that I added/changed: main.cpp, controller.cpp, controller.h, game.cpp, game.h, snake.c, smartSnake.cpp, smartSnake.h, renderer.cpp

How to play the game:

1. First step is to choose how many rounds do you want to play
2. Second step is to choose in every round if you want to play with a typical-manually played by the user snake or a smart snake
3. Third step: a) typical snake: you play the game as usual, b) smart snake: the smarty plays on its own until it appears with a red cube and that means that it is dead, so user must close window in order for the next round to begin.
4. Forth step: as it was already it’s implementation, if you want to stop the game premature you just close the window
5. Fifth step: if you want to see the statistics, then you must go to gamedata.txt that is in the build file

Loops, Functions, I/O

|  |  |
| --- | --- |
| CRITERIA | MEETS SPECIFICATIONS |
| The project reads data from a file and process the data, or the program writes data to a file. | Changes have taken place in files main.cpp, snake.cpp, game.cpp, renderer.cpp in order to write in a file the outcome of the game. Every time a snake eats, grows or dies , that action is written in file gamedata.txt that is stored in build file. Also, when the game ends, final score and size is written to gamedata.txt. It is an enhance functionality so that the gamer can observe game statistics. |
| The project accepts user input and processes the input. | In main, user is being asked to enter the number of rounds he/she wants to play. Every round statistics are being documented in gamedata.txt. Also in main, user is being asked to enter if he/she wants to play with a typical snake or a smart snake. |
| The project demonstrates an understanding of C++ functions and control structures. | Those criteria are being implemented in the whole code that 1) I have made adjustments, 2) I have created from scratch |

Object Oriented Programming

|  |  |
| --- | --- |
| CRITERIA | MEETS SPECIFICATIONS |
| The project uses Object Oriented Programming techniques. | The project code is organized already in classes and the things that I have added continue to be in that form. More specifically I have added class SmartSnake. |
| Classes follow an appropriate inheritance hierarchy. | Class SmartSnake (that is in file smartSnake.cpp, smartSnake.h) is a derived class from Class Snake. SmartSnake enhances Class Snake, in order , the objects of SmartSnake to have an intelligence. Objects SmartSnake, are acting on their “own”, simulating a smart behavior and moving towards to the correct direction that is pointed by the place of the food. Specifically function playSmart() implements that and also contents a lot of comments where I explain the algorithm. |
| Overloaded functions allow the same function to operate on different parameters. | In class controller I overloaded functions HandleInput and ChangeDirection, in order to be able to work both for standard Snakes and also for SmartSnakes:   * void HandleInput(bool &running, Snake &snake) const; * void HandleInput(bool &running, SmartSnake &snake) const; * void ChangeDirection(Snake &snake, Snake::Direction input, Snake::Direction opposite) const; * void ChangeDirection(SmartSnake &snake, Snake::Direction input, Snake::Direction opposite) const; |
| Classes use appropriate access specifiers for class members. | In order to be able to implement SmartSnake::playSmart() , I added a private member (which is SDL\_Point pointer) food (so that playSmart could see food’s coordinates and move the snake in the right direction). Food wasn’t necessary to be declared as pointer, but I declared it anyway, because I wanted to use raw pointers in my code (so to put in use all of my knowledge). In class SmartSnake I declared public methods : constructor, destructor, playSmart(). In class game I declared private members SmartSnake smarty, void UpdateSmartSnake() (it is a little different for smart snakes than the usual Update() method for standard snakes) . Also in class Controller, I added public function void HandleInput(bool &running, SmartSnake &snake) const and private function void ChangeDirection(SmartSnake &snake, Snake::Direction input, Snake::Direction opposite) const;. Both of whom have been mentioned above. |
| Classes abstract implementation details from their interfaces | As I mentioned before, I have implement a function in SmartSnake class , playSmart(). playSmart() contains the code that checks where how close or how far are the snake and the food and coordinates/moves the snake towards the correct direction. It is only called once in Game::Run and no one else ever calls or sees playSmart(). Implementation details are well hidden and everything works towards the correct direction. |