Advanced Programming Project Assignment

First, I would like to say that I have learned some very interesting and new things in C++. Some of those things would have come in handy when making the *project software engineering* and *chess game* project in the first bachelor.

I used some very useful Enums, Classes and Structs. I provided some examples below. This way I could subtract two position classes from each other. I made my own *position* class where the *operator overloading* was very practical.

|  |
| --- |
| enum Direction { *facingRight*, *facingLeft* }; enum FunctionCallTo { *DRAW*, *INPUT*, *SIMULATE* };  enum MoveScreen { *MOVE*, *NOMOVE*, *DEFAULT* }; |
| class Position { public:Position() = default;Position(float x, float y) : x(x), y(y) {}Position operator-(const Position& that) const {  Position newPos{};  newPos.x = that.x - this->x;  newPos.y = that.y - this->y;  return newPos;  }   float x;  float y; }; |

The way of dividing the game logic and game representation was very interesting. I liked the satisfying way in which the world can make entities without knowing anything about sfml (use of factories). Also, the singleton pattern was interesting because I didn’t know this method of keeping singularity of a class existed.

Something else that was new to me was clang-format, *CircleCI* and *Doxygen*. I never used them before but liked how clang-format made the code look structured. *Doxygen* took documenting code to a whole other level. That is something I am definitely going to use in upcoming projects.

At some points, this project was hard on me. There was a moment when I used sfml rectangles to define the entities, so had to redo a whole piece of algorithms. I am fan of how nice and clear sfml makes their documentation. It was very easy to get started with sfml.

Something else that proved itself useful were the *namespaces*. I used three namespaces being *ownView, ownModel and ownType*. That last one was used for the selfmade classes, structs and enums that I talked about earlier.

On the right is an example of my input files for the configuration of levels. The structure is very straight forward:

MOVE  
#################  
.................  
.................  
.................  
.................  
.................  
.................  
..............&##  
......###########  
......###########  
......###########  
......###########  
......###########  
......###########  
..........#######  
###.........#####  
#####.........###  
#######........##  
#########........  
###########......  
###########......  
###########......  
###########......  
###########......  
.................  
.................  
.................  
.................  
####.............  
########.........  
########.........  
########......###  
............#####  
..........#######  
........#########

* . represent open space
* # represent walls
* & represent the goal (meat boys girlfriend)
* At the top of the files there is a choice for MOVE or NOMOVE. MOVE will make the view of the level move.
* The basic width of a level is 17 tiles.

I would like to clarify some of the exception handling present in the code. The following exceptions are handled and printed to the user (the game doesn’t stop when an exception is present):

* The lack of files such as player sprites, background but also levels and the tileset. If missing, a goal is represented by a pink rectangle, a player by a red rectangle and a wall by a green rectangle. If for example level 1 is absent, the *inputparser* will try level 2 and so on.
* If MOVE or NOMOVE is missing from the level file, there will be movement.
* If no goal is given, a goal will be provided at any random location.
* If multiple goals are given, the first one will be kept but the following ones will be deleted.
* If the level exceeds the maximum amount of 17 tiles in width, the *inputparser* will “cut” the file and keep the first 17 tiles of every line and if an empty line was given, it will be ignored.
* If a character is wrong or a space was given, the character will be ignored.
* If the level is completely empty, the next level will be tried.
* If level 5 is completed, the game will go back to menu.