CIS 29, Winter 2021
Group #1:
- Yousef Helal
- Larry Shields
- Zachary Iguelmamene
- Ben Hoang
- Qing Mei Chen

Style Guidelines

1. All member function definitions should be placed outside of the class in the appropriate header file. The exceptions to this, are constructor initializer lists, and “one-liners”

2. Variables and functions should make use of camelCase

3. Class and Namespace names should make use of PascalCase

4. Global variables(including constants) should not be used

5. Class member variables should be declared either private or protected. To facilitate this requirement, getters and setters should be provided if there is a need

6. Function and Class beginning braces should be on the same line as prototype and class name, respectively

7. C-Style casting should not be used, and should instead be traded for static_cast, dynamic_cast, etc

8. C header files should not be made use of, and their C++ counterparts should be used instead.

9. All data written by files should be written in binary mode

10. smart_pointer and unique_pointer should be made use of as much as possible.

11. Any image displayed on the screen using SFML, should inherit from the Sprite class, which provides a number of methods for making dealing with sf::Sprite easier

12. If a class member function, or function otherwise do not alter any data values, they should be declared as *const* as often as possible

13. Header files should always have include guards to protect against redefinition.

14. The *auto* keyword should be used when dealing with iterators, as to avoid lengthy names

15. If you initialize memory on the heap, you are responsible for cleaning it up

16. Declaring two or more classes in the same header file should only be done if one inherits from the other, and one of the classes definitions is relatively short

17. Using *this* to access member variables is up to the person coding, and should depend on if the writer thinks the code will be more readable with its addition

18. Adding a file prefix to a file name should only be done using *Maps::filePrefix* or *Sprite::filePrefix* if inside a class that inherits from *Sprite*

19. If a particular part of a constructor is fairly long, an *init…()* member function should be created, and that should instead be called in the constructor.

STL Containers Used

- vector
- set
- map
- stack

C++ 11 Features Used

- Intiatilizer lists for constructors

- Making use of the *auto* keyword when dealing with iterators

- Making use of the *override* keyword when dealing with inheritance and virtual functions

- Making use of *make_unique* and *make_shared* when initializing addresses to points in memory

- Making use of the *default/delete* keyword when dealing with class definitions

- Making use of the *to_string()*

- Using initializer lists when creating objects of certain types(vectors, etc.)

<u>Other Requirements</u>

- The game makes use of SFML Libraries
- The game includes the reading of the keyboard (ex. MainTank.cpp, update() function)
- The game includes the sensing of a mouse (ex. Tank.cpp, fireBig() function)
- The game includes sound (ex. GameState.cpp, Gamestate() constructor)
- The game includes art work(the tank, bullet, etc. images, which are all stored in the ``data`` directory)
- The game includes an aspect of randomness(ex. EnemyTank.cpp, basicUpdate() function)
- The game includes a back door. It has two. If you press one ctrl key when pressing “Start Game”, you will be redirected to a testing ground where you may control another tank with the arrow keys. If you press two ctrl keys, you will be prompted to enter a level to “fast-forward” to. Code is located in MainMenuState.cpp updateButtons() function
- The game includes instruction/help, which can be accessed by pressing “Instructions”. Code is located in InstructionScreenState.cpp/h
- No 3rd party libraries other than SFML are used. SFML docs may be found here: https://www.sfml-dev.org/documentation/2.5.1/
- A multi-file application is used (as can be seen from file structure)
- Multiple classes are used (demonstrated in UML)
- Inheritance and polymorphism are used (demonstrated in UML)
- An “in-house” library is used(TankGraphics)
- Overloaded insertion operators are used(ex. Score.h/cpp)
- Exception handling is used(ex. ScoresScreenState.cpp, initHighScores() function)