PIC 10C Section 1 - Homework # 6 (due Sunday, May 8, by 11:59 pm)

You should upload each .cpp and/or .h file separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. The files you submit must compile and run under Visual Studio 2022 with all of the SFML libraries included when run in x86 Debug mode.

Read this carefully. You have two options:

1. Do the work individually as normal, submitting the usual Honesty.txt file.
2. Submit a single assignment as a group (max of 3/group — no exceptions). Only one member of the group can submit the assignment. In addition to the C++ files, you must submit a file "GroupWork.txt" that (i) explains how you divided your labour, (ii) summarizes of the work process of each member, and (iii) includes a brief reflection on the experience for each member. If you choose this path, any mulligans submitted for this assignment are handled individually (so a hypothetical party submitting a mulligan for this assignment may not receive help from their group members).

SFML GAME

In this homework, you will get to write a simple game — kind of like the dinosaur game on Chrome when the internet connection fails.

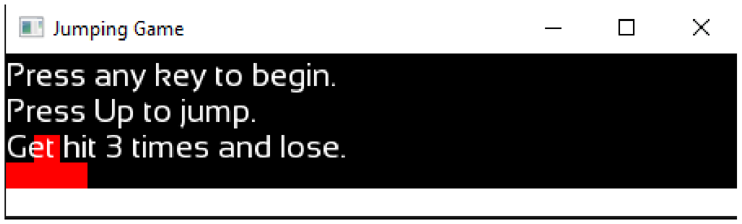
The accompanying **hw6.mp4** illustrates how the game behaves and detailed instructions will follow as to how to approach the problem.

Overview:

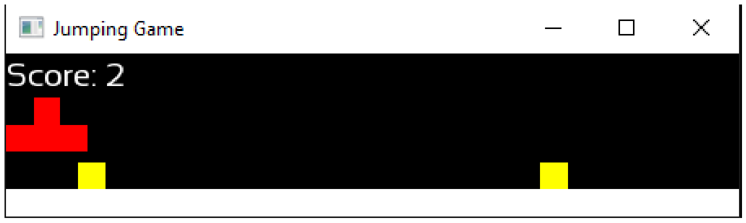
- The player's character is a red shape (square on top of rectangle) and the enemies/obstacles are yellow squares.
- The player is always at the left edge of the screen.
- The ground is represented by a solid white rectangle.

- At the start of the game, the user sees the white instructions:
  "Press any key to begin.
  Press Up to jump.
  Get hit 3 times and lose."
- If they press any key, the game begins.
- During the game, enemies are *randomly generated* (details follow).
- Enemies are yellow squares that move left along the ground that the player needs to jump over.
- During play, the score is always displayed in white at the top left in the form: "Score: [VALUE]" where [VALUE] is the number of enemies that have move past the left edge of the screen which they have successfully dodged.
- When the user jumps, a jump sound is made.
- If they touch an enemy, that counts as a hit and an explosion sound is made.
- After 3 hits, the game stops and beside their score is also the message "Game Over."

Remark: yes, there are a lot of moving parts to this homework — literally! But with careful object oriented design and with all the nice machinery of SFML it is very doable.



Press any key to begin.
Press Up to jump.
Get hit 3 times and lose.



Score: 2

Write **jump.h**, **jump.cpp**, and **main.cpp** to accomplish the game design. The font file and sound files are provided to you and you can assume they are in the same directory as the **.cpp** files you are writing.

This assignment has some absolute requirements and then some *strong hints* as to how you can accomplish the task. The absolute requirements must be followed. The hints are there to help you, but you may find alternative methods.

Absolute Requirements:

1. The Game window is named “Jumping Game.”
2. The Game instructions appear as described above.
3. The Game window is 450×100 pixels.
4. The ground and font are white, the character is red, and the enemies are yellow.
5. The ground spans the width of the window with a height of one-sixth of the window height.
6. The character is a square centred on top of a rectangle. The rectangle width is three times the square length, which is also one-sixth the window height.
7. The font is **sansation.ttf**.
8. The jump noise is given by **jump.wav**.
9. The hit noise is given by **hit.wav**.
10. Any key begins the game but only the up arrow allows for jumping.
11. The character’s pixel height h above the ground t seconds after their jump began is:
$$h(t) = \begin{cases} 8bt(T-t)/T^2, t \leq T \\ 0, T < t < T^* \end{cases}$$
where b the length of the square part of the character, $T = 1$ second and $T^* = 1.1$ seconds.
12. From the time of starting a jump until T^* seconds later, a subsequent command to jump is ignored.
13. The enemies always move along the ground, are square, and have a side length of one-sixth the window height.
14. The enemies move 2 pixels left every 0.01s.
15. Every 1.5 seconds, the Game may generate a new enemy with a probability of 50%.
16. The score increases every time the rightmost part of an enemy moves left of the leftmost part of the window.
17. If the player touches an enemy, the enemy that hit them disappears.
18. After 3 hits, the game ends.
19. Throughout the game and when the game is over, the displayed message is as described above.

Hints:

Write an **enemy** class. It can inherit from **sf::Shape**. It can...

- Store its own timer of **sf::Clock** type to track when it can move.
- Store some **static** members to dictate how far it can move in an update, etc.
- Have an **update_position** function that can be called upon by the **Game** to update the position if it is time.
- Provide useful accessor functions such as **get_x_min()** to return the minimum x-coordinate.

Write a **character** class. I wouldn't recommend inheritance here. It can...

- Store some **static constexpr** members for basic functionality like the T and T^* values, etc.
- Store members to track its bottom-left starting position and its current bottom-left position (which could change).
- Store a member to track if the object is in motion (jumping), call it **moving**, along with a timer.
- Store two **sf::RectangleShape**s to represent its bottom and top components.
- Store two **sf::Sound**s for the sound effects.
- Provide some accessor functions regarding its maximum x-position, etc.
- Have a **display** function that accepts an **sf::RenderWindow&**, making it draw both rectangles.
- Have a **path** member function to describe $h(t)$.
- Have an **update_position** function that can be called upon by the **Game** to update the position.
- Have functions **alive** (to check if they have not yet been hit 3 times), **hit_by** (to check if a given enemy is touching them), and **check_damage** (to check if they received damage and if so, to play a sound and track the hit).
- Have a **jump** function that only allows them to jump if they are not **moving**.

Write a **Game** class to manage all the other logic. The work can be neatly compartmentalized. For example, one **step** (a useful member function to have to advance the game) could call upon a series of functions like:

- **do_removals** (remove any enemies that are out of the window)
- **check_hits** (check if the player has been hit by an enemy and remove the enemy if they were hit)
- **move_enemies** (to move the enemies)
- **move_character** (to update the player's position)
- **make_enemies** (to possibly add more enemies)
- **manage_events** (to check if the user has pressed anything worth noting).

And after each **step**, there can be a call to **Game::display** that will ensure all the enemies, the player, and the text are displayed in the window.

Warning: do not attempt to run the code over a cloud service such as Dropbox as SFML may fail to load the resource files. It will still compile and run, though, just not behaving as you'd like.