Brief Description:

In this game, the player plays as an alien team of treasure hunters who are working to collect valuable treasure to bring back to their planet. They control the UFO that the aliens are traveling in and must avoid the falling meteors along the way. If a meteor hits the user, then the game is over.

As the game progresses, a greater number of meteors spawn, making it more challenging to reach the treasure. To reduce the number of meteors on the screen, the user also has the ability to shoot meteors in front of them using the “end” key. Doing so removes the meteor from the screen, although there is a timeout for shooting so the user cannot shoot rapidfire. The user moves using the ‘A’, ‘W’, ‘S’, ‘D’ keys.

Level 1: Start on this level, gems are worth 50 points each
Level 2: Reach this level when you hit 200 points, gems are worth 100 points each
Level 3: Reach this level when you hit 1000 points, gems are worth 150 points each
Level 4: Reach this level when you hit 3000 points, gems are worth 2000 points each

Documentation:

This game was written entirely using C++.

The screen is divided into an invisible 11 by 8 grid where meteors and gems can spawn. The user is able to move freely however.

File structure:

- GameTest.cpp
    - Contains all of the logic for updating and rendering the game
        - Misc:
            - Tracks the position of meteors on the screen using a 2D array (11 x 8) of CSimpleSprite pointers
            - The function to shift the meteors down by one is called every 4 seconds (set by a constant that’s easy to change) which was configured by using the deltaTime parameter in the update function
                - The total time the game has run for is tracked by adding the delta time each update and then compared to the time of last update for the meteors
                    - If the difference is greater than 4 seconds, the constant, then the meteor positions are updated
            - Vectors were used instead of arrays to reduce the amount of memory that had to be manually deleted and allocated
- Gem.cpp
    - Contains logic related to generating a new gem and the user obtaining a gem
        - void **setGem**(CSimpleSprite *gem, int &gemX, int &gemY)

            - This function sets the gem to a random section on the screen and updates gemX and gemY to hold the position of the gem
            - It works by first dividing the screen into sections for which the gem can spawn. Then, by calling a random number generator, we can randomly choose which section to place the gem and set it with the SetPosition function.
        - void **checkGetGem**(CSimpleSprite *gem, int &gemX, int &gemY, CSimpleSprite *player, float &score, int scoreIncrement)
            - This function checks if the player has reached the gem by checking which section the user is in, and comparing it with the section that the gem is in
            - It takes a gem, its position, a player, a reference to the score, and how much to increment said score by

- Levels.cpp
    - Contains logic related to the difficulty of the game as the level changes
        - std::vector<int> **generateArray**(int level)
            - This function generates an array of [METEOR_COLUMNS = 11] numbers based on the level, representing meteors
                - It generates a number from 0 to 4, where 0 is later interpreted as no meteor, and the meteors from 1-4 are just different sprites
                - The probability of getting a 1-4 is the same
                - The probabilities of getting a meteor for each level are set by the constants: *level1value*, *level2value*, *level3value*, *level4value*
- Meteors.cpp
    - Contains logic for generating a random line of meteors and also for moving the meteors down the screen
        - std::vector<CSimpleSprite *> **generateMeteors**(int level)
            - Generates an array of meteors (CSimpleSprite *) using the array generated from the function generateArray
        - void **updateMeteors**(std::vector<std::vector<CSimpleSprite *>> &allMeteors, int level)
            - This function shifts all of the meteors down and removes the bottom row of the meteors (freeing the memory), then adds the new row of generated meteors using the *generateMeteors* function
- Utils.cpp
    - Contains logic for utility functions such as for a game over condition or out of bounds
        - void **gameOver**(CSimpleSprite *player, std::vector<std::vector<CSimpleSprite *>> &allMeteors, bool &lose)'
            - Checks for game over by checking if the current position of the player is in one of the meteor occupied positions on the screen

        - bool **outOfBounds**(float x, float y)
            - Checks if the player is out of bounds by taking the user's current position
        - void **shoot**(CSimpleSprite *player, std::vector<std::vector<CSimpleSprite *>> &allMeteors)
            - Shoots a meteor straight in front of the player if one exists
                - This is done by checking for the first meteor in front of the player and removing it from the 2D array of meteors and replacing it with a nullptr

Credits for the media:

Meteors: https://opengameart.org/content/2d-asteroid-sprite
Ufo and background:
https://opengameart.org/content/simple-shoot-em-up-sprites-spaceship-starscape-ufo-0
Diamonds: https://opengameart.org/content/various-gem-stone-animations (Credit: "James (Lokiare) G. Holloway www.kelandlokgames.com")
Empty Gun Shot Sound: https://freesound.org/people/KlawyKogut/sounds/154934/