**Task 1:** Derivation of Easing Function

1. Start with the general form of the quadratic easing equation:

quadEaseOut = a \* (time^2) + b \* time + c

2. Plug points into two known points, starting point (0, begin) and ending point (duration, begin + change)

Starting Point:

quadEaseOut (0) = a \* (0^2) + b \* 0 + c = begin

c = begin

Ending Point:

quadEaseOut (duration) = a \* (duration^2) + b \* duration + c = begin + change

a \* (duration^2) + b \* duration + begin = begin + change

a \* (duration^2) + b \* duration = change

3. Solve system of equations to find values for a and b;

Substitute c from first equation into second equation:

a \* (duration^2) + b \* duration = change

a \* (duration^2) + b \* duration = change

a \* (duration^2) + b \* duration = change

a = change / (duration^2)

Substitute a into second equation

(change / (duration^2)) \* (duration^2) + b \* duration = change

b \* duration = change - (change / (duration^2)) \* (duration^2)

b \* duration = change - change

b = 0

4. Substitute ‘a’, ‘b’ and ‘c’ back into original equation:

quadEaseOut = a \* (time^2) + b \* time + c

quadEaseOut = (change / (duration^2)) \* (time^2) + 0 \* time + begin

quadEaseOut = (change / (duration^2)) \* (time^2) + begin

Simplify further:

quadEaseOut = (-(change / (duration \* duration)) \* (time \* time) + begin

**Task 2:** Comparison with other potential easing functions

**Linear Easing**: Linear easing provides a constant rate of change between the starting and ending values. It has a simple equation that linearly interpolates between the values based on the current time.

sf::Vector2f EaseLinear(sf::Vector2f begin, sf::Vector2f change, float duration, float time)

{

float t = time / duration;

sf::Vector2f linearEase = (change \* t) + begin;

return linearEase;

}

Linear easing produces a linear transition without any acceleration or deceleration. It might be suitable for scenarios where a constant rate of change is desired.

(Duim, *Easing functions for JavaScript - animation tool: Spicy Yoghurt*)

**EaseOutCubic**: Ease Out Cubic is another easing function that applies cubic easing to the values. It provides a more pronounced easing effect compared to quadratic easing.

sf::Vector2f EaseOutCubic(sf::Vector2f begin, sf::Vector2f change, float duration, float time)

{

float t = time / duration - 1.0f;

sf::Vector2f cubicEaseOut = (change \* (t \* t \* t + 1.0f) + begin);

return cubicEaseOut;

}

The cubic easing equation used in this function gives a smoother and more gradual easing effect compared to quadratic easing. (Duim, *Easing functions for JavaScript - animation tool: Spicy Yoghurt*)

**Task 3:** Why is EaseOutQuad a good choice in this case?

The choice of an easing function depends on the desired animation effect and the specific application. However, EaseOutQuad can be a good choice for the following reasons:

* Smooth transition: EaseOutQuad provides a smooth and gradual easing effect. It starts quickly and slows down towards the end, creating a natural and visually pleasing animation. (Paullewis, *The basics of easing*)
* Simple implementation: The EaseOutQuad equation is relatively simple and easy to understand and implement. It requires basic arithmetic operations and does not involve complex calculations or external dependencies. (Duim, *Easing functions for JavaScript - animation tool: Spicy Yoghurt*)

**Sources**

Paullewis (no date) *The basics of easing*, *web.dev*. Available at: <https://web.dev/the-basics-of-easing/> (Accessed: June 2023).

Duim, C. (no date) *Easing functions for JavaScript - animation tool: Spicy Yoghurt*, *Easing Functions for JavaScript - Animation Tool | Spicy Yoghurt*. Available at: <https://spicyyoghurt.com/tools/easing-functions> (Accessed: 27 June 2023).