**Introduction**

In this project, a PID-controlled Dino Game has been implemented using JavaScript, CSS and the PID (Proportional-Integral-Derivative) algorithm. The game, reminiscent of the popular Chrome browser game, challenges players to control a dinosaur (represented by the red box) to avoid obstacles (represented by the green box) by jumping over them. The integration of the PID algorithm adjusts the dinosaur's jumping behaviour based on the distance to the upcoming obstacle.

On the website hosting the game, there’s the option to tune the PID parameters to customize the dinosaur's jumping behaviour. Additionally, there is a graph on the website that shows the PID output in function of time according to the parameters tuned. Also, a Matlab Simulation is linked on the site.

There’s an option to play the game without the PID control, providing a comparison of the gameplay with and without the algorithm. This project demonstrates the application of PID control in the Popular Chrome Dino Game.

**How The Game works without PID control**

The Dino game code consists of two main elements - the red object and the moving green object. When the “up” arrow is clicked, an event listener triggers the red object to jump. The green object moves towards the red object using a CSS animation. A function called checkDead() is responsible for detecting if the two objects collide with one another. If a collision occurs, checkDead() stops the game and the score counter.

**How the PID Algorithm is Implemented**

The PID (Proportional-Integral-Derivative) algorithm is implemented in the Dino game using a function called "pidcontroller()". This function takes as input the desired value and the three parameters: P (Proportional), I (Integral), and D (Derivative). These parameters are used to calculate the output value, which is then used to determine when the red object jumps.

The P parameter determines the proportion of the error that contributes to the control output. The I parameter considers the accumulation of past errors to adjust the control output. The D parameter accounts for the rate of change of the error.

In the Dino game, the "pidcontroller()" function returns a function called “pidControllerFunction()” that returns the output value, the reason for returning a function in the first part is to calculate an initial output value and then call the returned function “pidControllerFunction()” iteratively to make corrections.

**How the Game works with PID control**

The "checkDead()" function transforms into a PID-controlled version. In this new version, the event listener for the "Up" arrow key is disabled. Instead of directly triggering the jump, the jump is now determined based on a value called "jumpDist," which represents the distance between the red object and the green object.

The PID portion of the code is responsible for taking the desired jumping distance value of 45px and adjusting the "jumpDist" value to approach the desired value factoring in the timing and the dimensions (height and width) of both objects, ensuring that the red object jumps in time.

**Considerations and Restrictions**

In the Dino game, several considerations and restrictions impact the behavior of the "checkDead()" function and the animation of the green object. The "checkDead()" function runs at a specific rate, which affects the frequency at which collisions between the objects are detected. On the other hand, the animation of the green object takes 2 seconds to complete, Due to the timing differences, there is a possibility that some distance points of the green object will not be detected by the "checkDead()" function. As a result, the "jumpDist" value becomes an interval rather than a precise value. This interval depending on the frequency of the "checkDead()" function in this case is 4px. For example, if the desired "jumpDist" is 45px, the interval might be set between 41px and 45px.

Another important consideration is the selection of optimal PID parameters. In this context, the parameters (P: 0.1, I: 0.5, D: 0.1) have been identified as optimal. While other parameter combinations may also yield an output equal to the desired value, the timing aspect plays a crucial role. A shorter settling time is crucial for the game's responsiveness. For instance, the parameters (P: 0.1, I: 0.1, D: 0.1) result in a larger settling time compared to (P: 0.1, I: 0.5, D: 0.1), making the second more optimal due to its quicker response.