Fuzzy Control for Racing Games

Coursework Report for CE1184a/k
University of Abertay, Dundee

Sarah Herzog

18 Jan 2013

Introduction

In any game possessing non-player entities, in-game control of these entities is an important factor in player immersion. If non-player entities do not act in ways the player expects, enjoyment of the game is reduced. It follows that controlling these entities in a smooth, believable way is of vital concern to game developers.

This project and report examines the specific genre of racing games and the control of non-player cars in games of this genre. This problem is not well served by the discrete finite state machine traditionally seen in game artificial intelligence, as this solution would cause the cars to move in stuttering steps. Changing between discrete values for velocity is not something a human would do, and consequently looks very unbelievable to players. It also makes it more difficult to control the car, causing overshooting of the target location, since fine adjustments are not possible.

Thankfully, researcher Lofti Zadeh at University of California Berkeley introduced the concept of fuzzy logic in 1965 (Bourg and Seemann, 2004: 188). This allows game developers to build a finite state machine based on fuzzy sets with partial set membership rather than discrete, binary set membership. This technique is ideally suited for racing game non-player entity controllers (Bourg and Seemann, 2004L 190-191), and is thus the focus of this report.

To fully explore the application of fuzzy logic in games, a fuzzy logic controller was designed and implemented in a simple racing game proof of concept. In the game, a car (represented by a red circle) attempts to stay on the road (represented by a white line). Constant forward motion is assumed for simplicity, so only the x component of velocity for both the road and car is considered. The controller for the car takes two inputs: the relative position of the road with respect to the car, and the relative velocity of the road with respect to the car. The controller then recommends a change in absolute velocity of the car.

One difficulty of fuzzy logic controllers is the large amount of tuning needed to achieve an ideal controller (Bourg and Seemann, 2004: 210). Such a controller has so many degrees of freedom that it can be difficult for a human to intuitively choose a good set of parameters, so some trial and error tuning is required. Even worse, changing even one variable in the car or track could cause a drastic change in performance (for example, decreasing the maximum acceleration achievable by the car, or increasing the curviness of the track).

Because of this, an alternative method for tuning such a controller was investigated. Genetic algorithms seemed ideal for this task, as they allow exploration of the solution space in a targeted fashion while providing an acceptable solution at any stage of the process, allowing for early termination as desired by the developer (Bourg and Seemann, 2004: 317-318). Using such a method to tune a controller could ultimately save the developer a large amount of time, especially when the game has a large variety of cars or tracks.

The remainder of this report details the specific application used to investigate fuzzy logic and genetic algorithms as they apply to racing games. The methodology section will cover all design choices including universe of discourse, membership function shape and locations, rules, and limitations on car acceleration and velocity. It will also cover software design choices such as language and program structure. The testing procedure section will discuss monitoring procedures for the controller and how results were obtained. It will also discuss the automated track design. The results section will present detailed statistics for both the manually tuned controller and the controller produced through genetic algorithms. Finally, the conclusions section contains reflections on the project, lessons learned, and recommendations for future investigation.

Methodology

The fuzzy logic controller application required a massive number of design decisions. In general, these decisions were informed by key goals. First, since much of a fuzzy logic controller's versatility comes from it's ability to be tuned, flexibility was held vitally important. Second, a focus on the project at hand was maintained, so some features had to be discarded in order to keep the project within scope. Third, a full demonstration of fuzzy controllers and their inner workings was desired, so the application shows as much of what is going on internally as possible.

The rest of this section describes how these three goals informed design decisions throughout the project development.

Fuzzy Controller Design

Project specifications dictated that the controller should accept two input variables and provide one output variable. For input variables, the controller accepts the relative position and velocity of the road with respect to the car. These inputs allow the controller to handle cases where the car is already moving toward the road, or situations where the road may be close by but moving swiftly away.

For output, the controller recommends a change in absolute velocity of the car - in effect, acceleration. This output was chosen because it allows the car's maximum acceleration to be limited. This results in a more realistic representation of the problem, where the results of the controller are limited by the capabilities of the car.

Universe of Discourse

The universe of discourse was chosen as -600 to 600 units for all variables (position, velocity, and action). This was primarily for ease of programming, since the width of the HTML5 canvas where data would be displayed was chosen as 1200. No translation from game position to canvas position would be necessary as they are one and the same.

Number of Membership Functions

At least two membership functions are necessary for a control solution to this problem, since a distinction between left and right must be made. However, more granularity and flexibility was desired, so a total of five membership functions were used for each input. For the output, nine membership functions were used. This allowed a more granular set of responses, given each combination of inputs, as will be shown in the rules section.

Membership Function Shape

Again with the goal of flexibility in mind, membership functions were designed to allow as many shapes to be tuned as possible, while handling all shapes in an identical fashion within the code. A modified trapezoidal style membership function was chosen for all sets. This allows both trapezoids and triangles to be created, since a triangle is simply a trapezoid with it's two top corners at the same point (demonstrated in Figure 1 below).

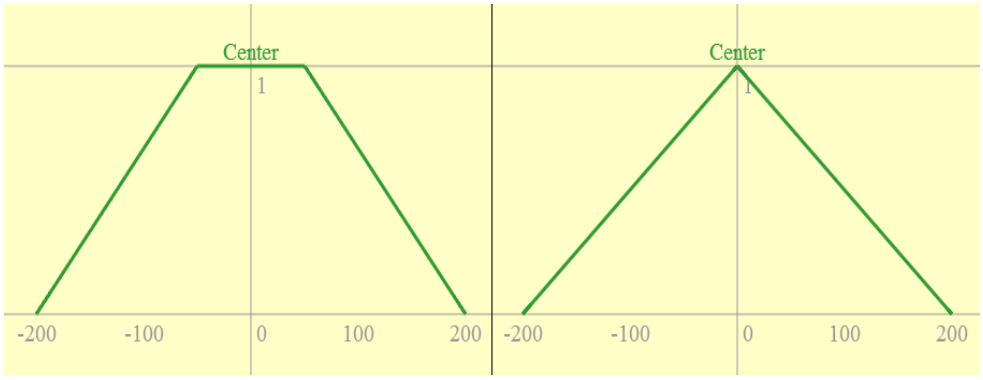

Figure 1: *Figure demonstrating a trapezoid with left and right peak points on the same coordinates, as shown in the developed application's interface.*

In addition, an attribute dubbed "curviness" was added (demonstrated in Figure 2 below). This attribute causes the slanted sides of the trapezoid to become curvy. The idea was to additionally allow a shape similar to the gaussian or bell curve.

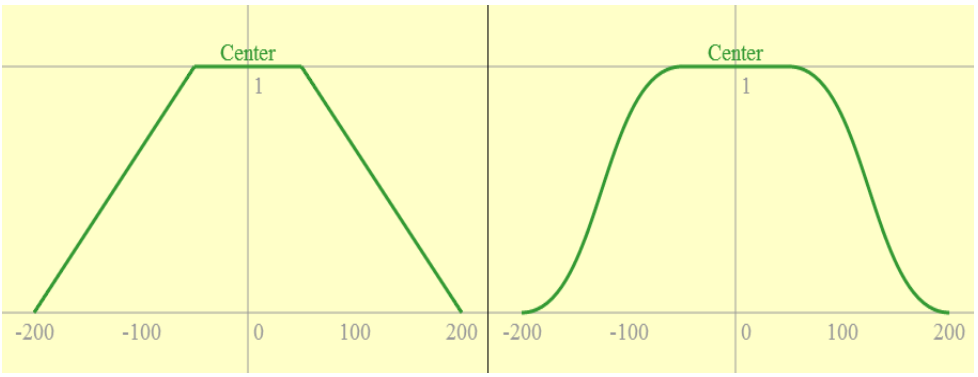

Figure 2: *Figure demonstrating a trapezoid with "curvy" sides, as shown in the developed application's interface.*

After some research, a type of cubic spline called a Bézier curve was chosen as the implementation for these curves. This choice was made easy due to the native support for Bézier curves in the HTML5 canvas (Buckler, 2011). Cubic Bézier curves are specified using four points: two endpoints and two control points. These four points are combined in a parametric equation to specify the curve (see Figure 3 below).

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1].$$

Figure 3: *Cubic Bézier parametric equation (Wolfram MathWorld, 2013).*

The endpoints in the controller's membership functions are simply the endpoints of the side of the trapezoid. For example, the left side of the trapezoid has endpoints P_0 at y = 0 and P_3 at y = 1. The control points are set by default on top of their corresponding end point, so $P_1 = P_0$ and $P_2 = P_3$. The x values for the control points are then modified by a "curviness factor" set by the user, a value between 0 and 10. This is scaled by the difference between the two endpoint's x values, then either added or subtracted to the control point as appropriate. As an example, the x coordinate for the bottom control point of the left side of the trapezoid is $P_{1x} = P_{0x}$ + curviness factor * (P_{3x} - P_{0x}).

Using this method, it is possible to obtain y given x, or x given y. However, in practice, there was much difficulty obtaining an x given y. Using Bézier curves requires solving a cubic equation, and while this works fine going in the x to y direction, problems are encountered going from y to x. Cubic solutions can contain imaginary numbers, and in particular a solution where the independent variable is between one and zero will always involve complex math (based on equations given by Wolfram MathWorld, 2013). This is always the case when going from y to x, since y is always between one and zero when on the curve. Unfortunately, the chosen language for this project (JavaScript) has no native support for complex numbers, and writing a custom class to handle complex math was outside the scope of this project. Because of this, curviness is not taken into consideration for output membership functions, but is available for input functions in manual tuning. It was removed from genetic algorithm tuning due to longer computation times.

Ultimately, the curviness feature was in all likelihood not worth the time invested in it. It would be interesting to do more testing and determine if changing the curviness actually has much effect on the resulting controller's performance.

Membership Function Location

In the manually tuned controller, membership functions were equally spaced across the universe of discourse to assure that at any given point a raw input will be part of at least two fuzzy sets, excepting the extreme right and left functions. Without this, the car tends to oscillate across the road rather than settling at the center. This also insures a smooth behaviour rather than having any sudden changes from one velocity to another. Membership functions for all variables can be found in Figures 4-6 below.

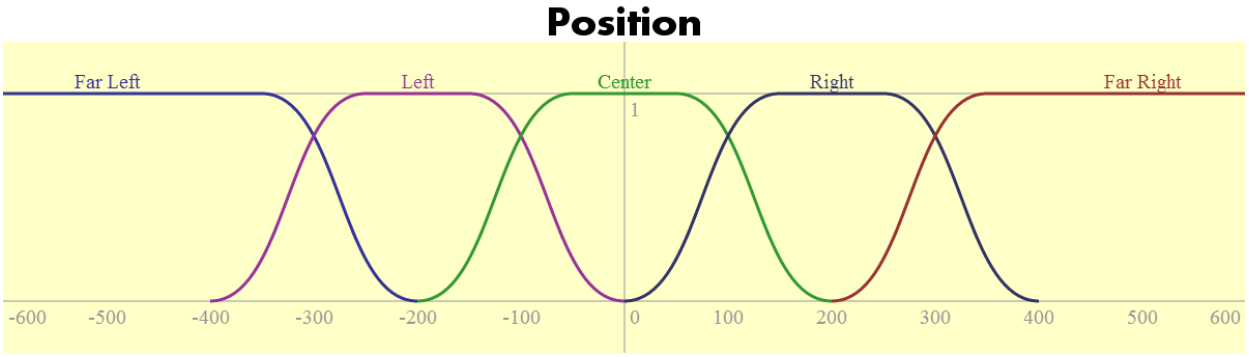

Figure 4: *Position (input) membership functions for the manually tuned controller.*

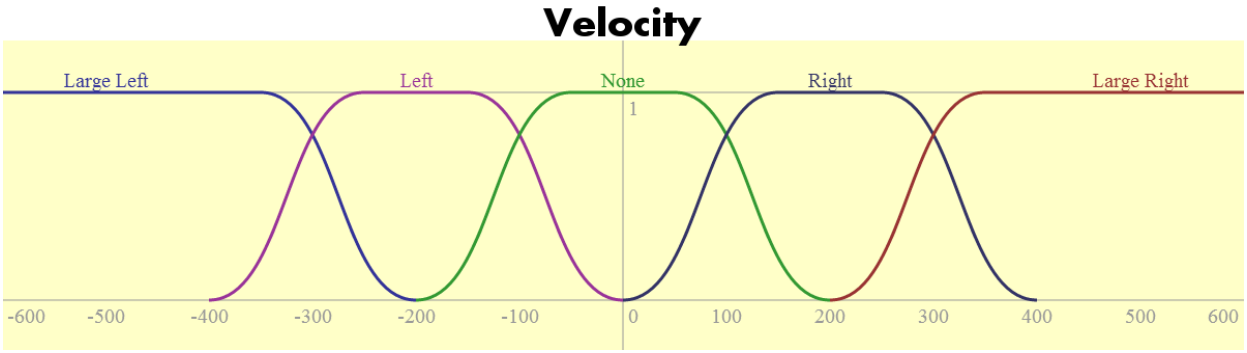

Figure 5: *Velocity (input) membership functions for the manually tuned controller.*

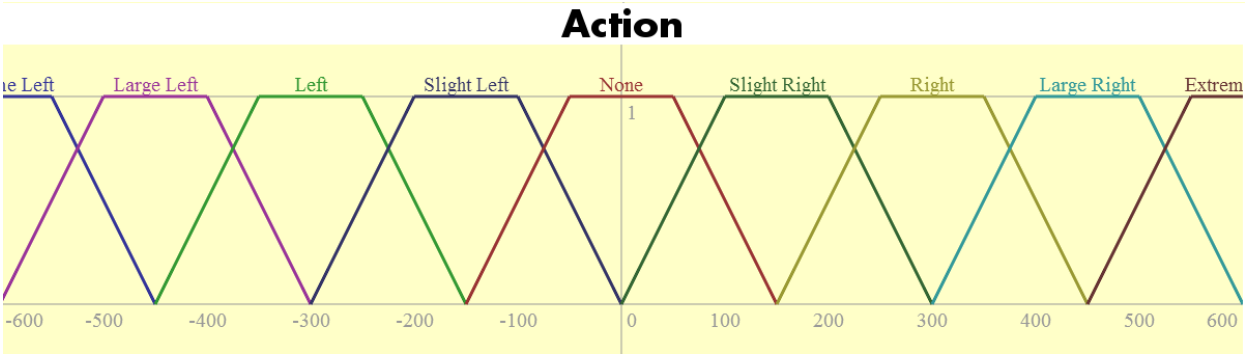

Figure 6: *Action (output) membership functions for the manually tuned controller.*

Rules

The rules were determined using a fuzzy associative map, shown in Figure 7 below. A mirrored approach was taken, with extreme responses only used in the most extreme combination of inputs, while more reserved responses are common along the line of symmetry (diagonal from lower left to upper right). Notice a diagonal mirror effect, with left-oriented responses in the upper left and right-oriented responses in the lower right.

		Line Location				
		Far Left	**Left**	**Center**	**Right**	**Far Right**
Velocity of Line	**Large Left**	Extreme Left	Large Left	Left	Slight Left	None
	Left	Large Right	Left	Slight Left	None	Slight Right
	None	Left	Slight Left	None	Slight Right	Right
	Right	Slight Left	None	Slight Right	Right	Large Right
	Large Right	None	Slight Right	Right	Large Right	Extreme Right

Figure 7: *Fuzzy associative map showing rules for the manually tuned controller*

This ruleset provides a smooth response to any situation. Combined with a large number of outputs and overlapping membership functions, the specified controller reacts with fine or large adjustments properly for each combination of inputs. The resulting surface map showing the controller's response to different situations can be seen in Figure 8 below.

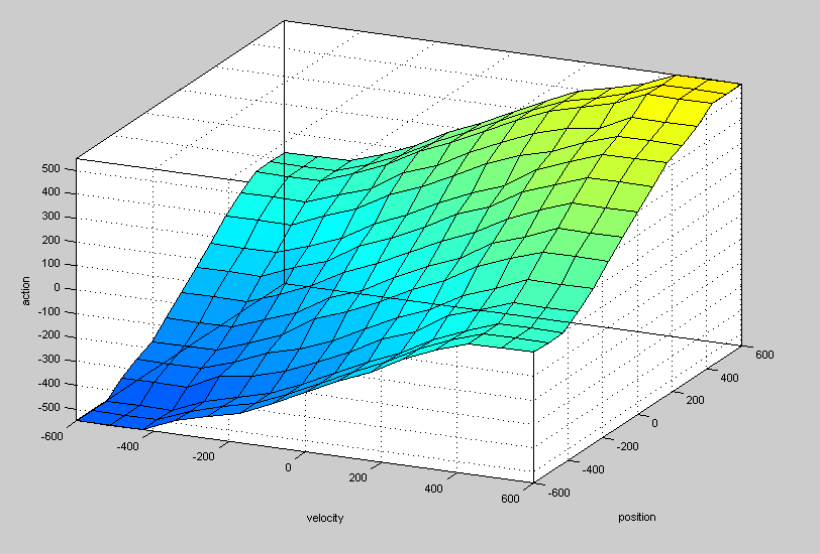

FIgure 8: *Surface map showing the manually tuned controller's response against input combinations.*

Software Design: Proof of Concept

The proof of concept application was designed to be as simple as possible while still demonstrating the use of the fuzzy controller. For this purpose, the application is graphically simplistic, representing the road as a white line and the car as a red circle. Different camera orientations were experimented with, such as centering the camera on the car or road, but ultimately the controller performance was clearest when the camera was stationary and centered at position zero.

The proof of concept program structure was very simple, but attempted to maintain an object oriented approach. The controller, car, and road were each isolated into their own class. Results are tracked and processed in a separate class, though in retrospect it may have made more sense to include this in the controller class. These classes are then updated by a manager class, such as the game class. This approach proved very useful when the genetic algorithm section of the project was coded. It allowed a large array of each game object to be created and run simultaneously by the genetic algorithm class.

As per the project specification, encapsulation of the AI was held paramount. Another AI object using the same interface could replace the fuzzy logic controller and the proof of concept game would not know the difference. This also allows controllers with different settings to be swapped with ease.

Platform and Language

Javascript and the HTML5 canvas were chosen as the platform for this project, primarily due to their potential for improving controller flexibility. This allowed a simple controller editing interface to be built with ease, making instantaneous changes in the controller possible without having to wait for compilation. It also encouraged a focus on the application itself, avoiding the time sinks of syntax issues, memory management, and type errors.

Game Design Decisions

To make the problem more realistic, limits were placed both on the maximum velocity the car can obtain, and also on the maximum acceleration of the car. In a racing game, different cars often have different handling, acceleration, and top speeds. Placing these limits on the controller will consequently yield results that are more relevant to the controller's potential use in a real racing game.

Fuzzification

Moving from raw inputs to fuzzy set values is achieved using a piecewise definition of the membership function. This uses the four points of the trapezoid to define the domain of each function. Below the left base point and above the right base point the membership is zero. Between the left base point and left peak point, and between the right peak point and right base point, the membership is between 0 and 1. This is the sloped portion of the graph. If the curviness of the slope is zero, the function is linear and the result can be easily determined via linear interpolation. If the side has a curviness value, the cubic Bézier described above must be used to determine the fuzzy membership value. Finally, if the raw input is between the left peak point and the right peak point, the membership is 1.

Rule Interpretation

Rules in the proof of concept application are represented as a multidimensional array in which the first index is the position, the second is the velocity, and the entry at that location is the recommended action for that combination. A multidimensional array worked well in this case since there is a rule for every combination of inputs. It was also planned in order to make genetic algorithm modification of rules easier if desired, though ultimately it was decided not to modify rules via the genetic algorithm.

Output membership is determined using several steps. First, inputs are fuzzified using the method described in the previous section. A zeroed array for recommended actions (output) is also created. Second, each possible combination of inputs is looped through. For each combination, the input and output are passed into the multidimensional array of rules to

determine the recommended action. The inputs are then combined using an AND (minimum) function, then combined with any existing membership in that output using an OR (maximum) function.

Once this process is complete, a set of fuzzy output values is ready for defuzzification.

Defuzzification

Defuzzification for the proof of concept application uses a Center of Maximums method. This method allows reasonable accuracy while being relatively simple to implement. However, inaccuracies are introduced when membership functions overlap considerably, or are significantly lopsided (non-symmetric). For the scope of this project, however, these downsides are outweighed by the simplicity of the method.

Research showed that there are several methods known as Center of Maximums, all similar but with some small differences (National Instruments, 2010). The method used in this project is similar to the one discussed by Dr. King in his lecture of Fuzzy Logic (King, 2012). However, in Dr. King's method, overlapping functions may lose some data since results are examined as if plotted all on the same graph - a function may overlap and appear to have less area than it might originally have had if examined alone. The version used in the proof of concept application was easier to implement, as it examines each action membership function alone. Consequently, it may produce different results and doesn't take into account interaction between output membership functions caused by function overlap.

Each membership function is defuzzified in a very similar way to the fuzzification process described above, except this result yields both a value and an area (weighting). Each function is examined in a piecewise fashion given the input fuzzy value. Inputs of zero yield a result of zero. Inputs of one yield a crisp output equal to the average of the two peak point values for the trapezoid. The weighted area is equal to the difference between the peak points (multiplied by one, the height of the trapezoid). Inputs between zero and one fall on the edge lines. Since the cubic function is difficult to calculate in this direction, no curviness is allowed for output functions, and so these calculations are simply linear. Once the x coordinates are determined for both sides, they are averaged to obtain the defuzzified value. The weighting is then determined by their difference multiplied by the input fuzzy value.

Finally, once all values and weightings are determined, the weighted average of these values is calculated to obtain the final crisp action value.

Genetic Algorithm: Design and Implementation

Though it was being used to circumvent some of the tuning for the fuzzy controller, the genetic algorithm required some tuning as well. There were several important pieces to its execution

that had to be designed, including population size and randomisation, simulation and fitness evaluation, and selection, crossover, and mutation methods. Each of these will be discussed in turn.

Population Size

The population size was set to one thousand based on literature recommendations, which claimed most genetic algorithms used hundreds to thousands of solutions in each generation (Obitko, 1998). The size is limited by the processing capabilities of the computer running the simulation and the amount of time the user is willing to wait for a solution. In general, a larger size allows for a more diverse population and better exploration of the solution space, but there are diminishing returns for large sizes.

Population Randomisation

The method for population randomisation went through several iterations. At first, a controller was randomised by looking at each membership function individually. The left base point was determined randomly, then the left peak point with the constraint of being equal to or greater than the left base point, and so on. However, this resulted in skewed populations leaning heavily to the right.

A second approach was developed where again each membership was examined individually. A set of four points were determined randomly, then sorted, and finally assigned to each point of the trapezoid in order. This provided a far superior randomisation, but performance was terrible precisely because of this extreme randomness.

In the end, a “seeded” approach was taken, so that solutions, while still somewhat random, were likely to have decent performance. Instead of examining membership functions in a vacuum, each variable is viewed as a whole. A large set of points (enough for every defining point in each membership function) is generated and sorted. These points are then assigned to the membership functions in a logical order, so the left function is on the left, and so forth. The resulting functions are much more likely to behave effectively as a controller.

Rules are not randomised in the current genetic algorithm approach, though the program structure does allow this easily. Randomising rules would cause very erratic behaviour in the simulation. It is likely that the designed ruleset is a good choice and the variation of membership functions is sufficient for tuning purposes without varying the rules as well.

Simulation

The simulation is almost identical to the game class used to demonstrate the fuzzy controller. However, the number of operations per second is increased drastically, and there is no drawing to screen during the simulation. Additionally, rather than having one of each game object, the simulation runs an array of one thousand of them. Due to the game's object oriented design, however, this was quite simple to implement.

Fitness Evaluation

Fitness of a solution is evaluated based on its performance in the simulation. For this purpose, the average distance between the car and the road is considered. The idea is to minimise this value. However, most selection algorithms rely on a high fitness value to indicate a better solution, rather than a low value as is the case with the average distance. Consequently, this value must be inverted in order to be used. To do this, it is subtracted from the worst average distance in the population.

Once this modified value is obtained, it is normalised by dividing it by the sum of all the fitness values. The solutions are then ordered from highest normalised fitness value to lowest. After this, the accumulated normalised fitness value for each solution is determined by adding its normalised fitness value to the sum of all those before it in order. This will facilitate the selection process in the next section.

Selection

This project uses a roulette-wheel style selection process (Obitko, 1998). This type of process ensures that the population does not stagnate by selecting some less-fit solutions, though a higher fitness does result in a higher chance for selection.

This process relies on random selection, in which more fit solutions are more likely to be selected. First, a number is generated between 0 and 1. The algorithm loops through the array of solutions, which were ordered and prepared in the previous section. The result of that ordering is that solutions are ranked in ascending order, with the final solution having an accumulated fitness of one, and the larger the difference between solutions (the larger the section of the roulette-wheel, so to speak) the better that solution performed. Better solutions are also ranked earliest in the order. As soon as a solution is found with an accumulated normalised fitness value less than or equal to the randomly generated number, that solution is selected and the process begins again.

Crossover

The uniform crossover method (as described by Obitko, 1998) was chosen, primarily due to the nature of the representation of each solution's genetics. While several methods were experimented with, in the end, a method similar to the randomisation method was used for crossover. Each individual member function is examined, and each point is chosen from one of the two parents (this choice is determined randomly). These points are placed in an array, sorted, and then placed into the membership function in order. In this way, membership functions maintain the correct shape but still pull randomly from both parents.

Mutation

This project uses the uniform mutation method, due to the integer nature of the data (Obitko, 1998). The mutation chance was set to 0.05%, which results in several mutations per generation. After crossover, each new offspring has every point of each membership function examined. A random number between zero and one is generated, and if it exceeds the mutation chance, that point mutates. The new point is a random point bound by the universe of discourse and by nearby points in the membership function. For example, a left base point is bounded below by -600 and above by the left peak point.

Testing Procedure

Testing for the controller and genetic algorithms involves several key points. These include the automated track, manual track control, and results reporting.

Automated Track

Testing for the controller is done primarily by an automated track. The track uses the artificial intelligence method of pattern movement (as discussed in Bourg and Seemann, 2004: 27-51), and uses a set of instructions where a velocity and end position are defined. This allows for a fairly complex set of motions to be defined. The track pattern chosen was designed to exercise as much of the controller as possible, using many different velocities and changes in velocities to do so. The track includes both acceleration and deceleration in a single direction, rapid directional changes at a single speed, and rapid directional changes at varying speeds. A chart showing the track's dictated road position over time can be seen in Figure 9 below.

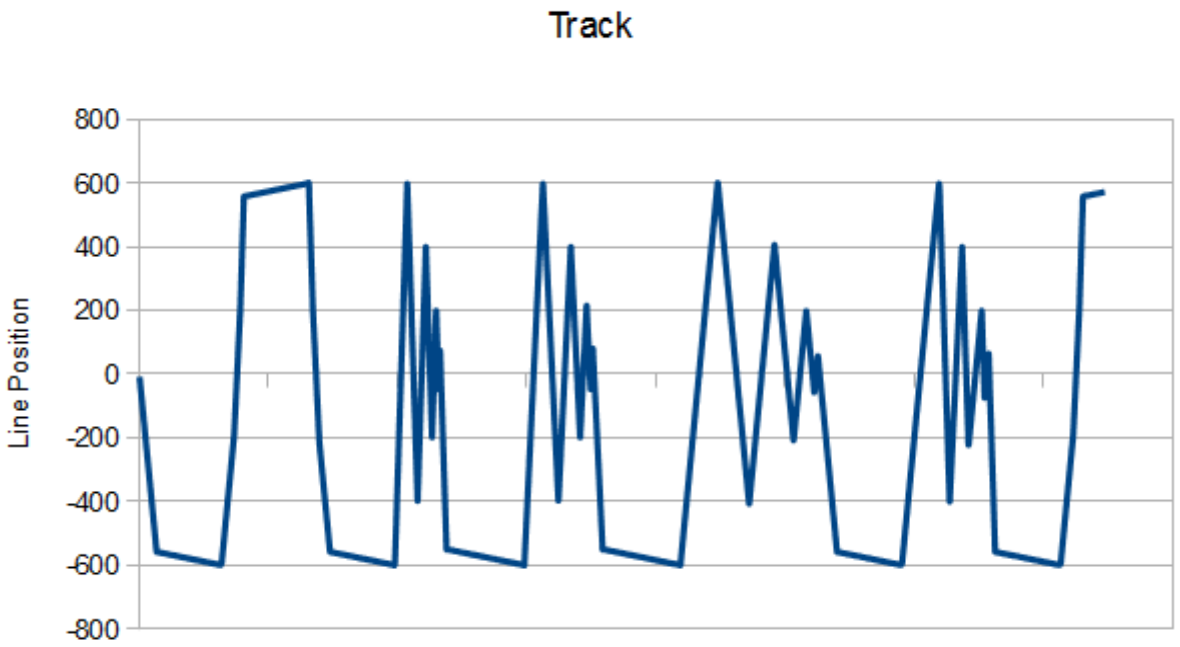

FIgure 9: *Chart showing the road position dictated by the track over time.*

Manual Track Control

In addition to the automated track, manual track control allows the user to exercise the controller manually. For both the automated and manual track, three speeds are used. The user can switch between speeds either using the number keys or up and down arrows, and controls the track itself using the left and right arrows.

Results Reporting

In order to actually measure the effectiveness of the controller, data must be recorded during the controller's operation and processed to determine several important statistics. During operation, the results object tracks position and velocity of both the car and the road. Afterward, it processes this information to determine the sum, mean, and median distances from the line, as well as the standard deviation. These statistics allow evaluation of the controller performance. The results object also determines the percentage of time spent in each of the variable sets. This helps determine whether the track did a satisfactory job of exercising the controller.

Results

Using the testing procedures described in the previous section, measurements of the fuzzy controller and genetic algorithm tuning were obtained. These include statistics for the manually tuned controller, effectiveness of the controller produced by the genetic algorithm tuner, and an evaluation of the automated track effectiveness.

Effectiveness of Manually Tuned Controller

The manually tuned controller (described in detail in the previous sections) obtained a satisfactory performance. Visually, from a player perspective, the car moved smoothly and naturally. It could not always keep up with the line due to a set maximum velocity and acceleration, so the average distance is not expected to be close to zero.

The result recorder tracks raw data and then processes it to obtain helpful statistics.
When processed, this data yields a mean deviation from the track of around 35.6 units. This seems a little high, but the median is only 7.6 units, with a standard deviation of 62.37 units. The histogram in Figure 10 below shows that this is due to most of the data being clustered close to zero, with a steady tail of data stretching all the way out into the 400 range.

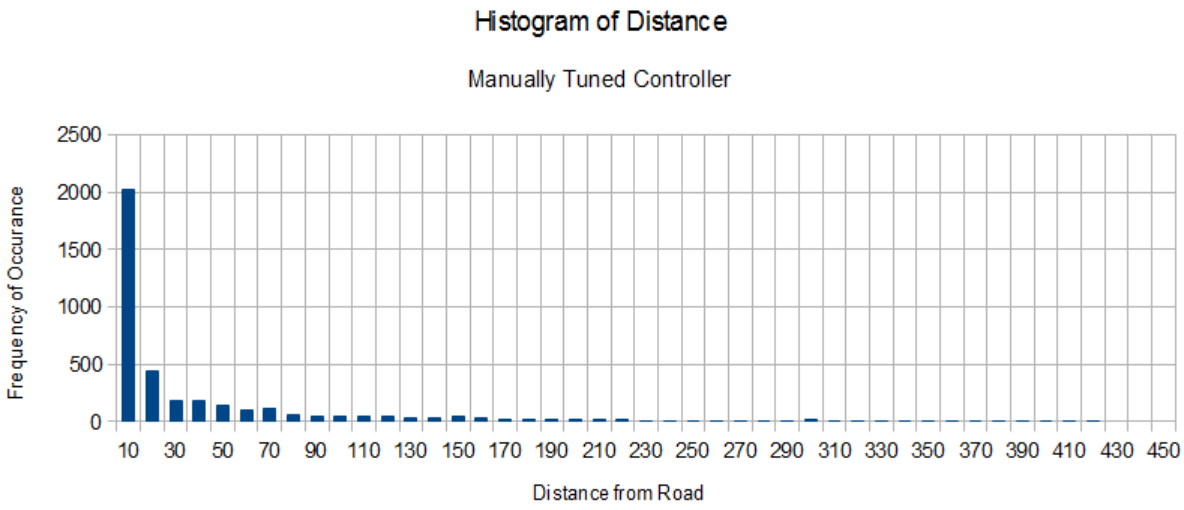

FIgure 10: *Chart showing frequency of distances, in increments of ten, for the manually tuned controller*

Ultimately, it is difficult to say whether this performance is good or bad without anything to compare it to. The most important question, “Does the movement look natural?”, is answered without any data at all. From this player’s judgement, it does.

Effectiveness of Genetic Algorithm Tuning

With the manually tuned controller as a baseline, a comparison can be made to the controller tuned using genetic algorithms.

After over five hundred generations, each with one thousand controllers, the algorithm found the "ideal" controller depicted by the membership functions in Figures 11-13.

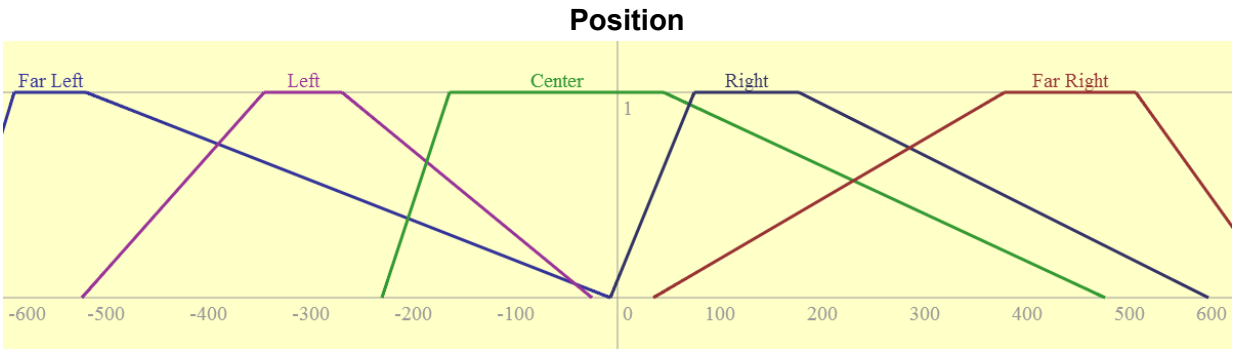

Figure 11: *Position (input) membership functions for the genetically tuned controller.*

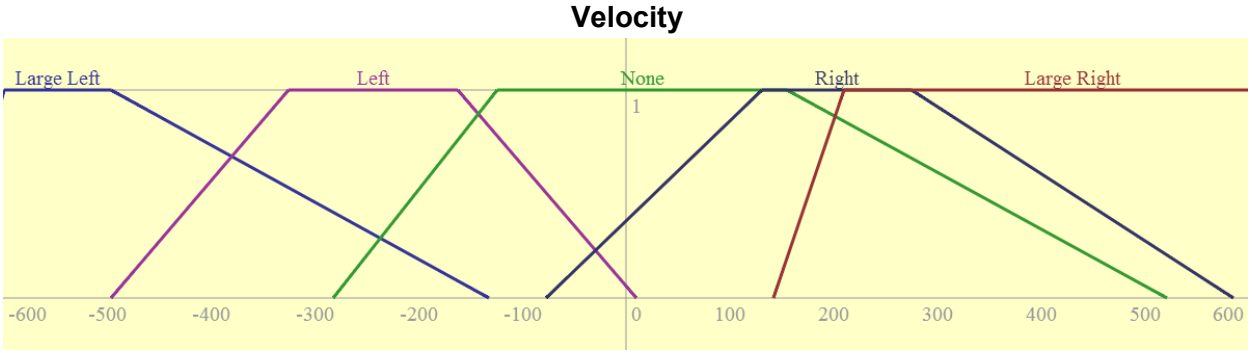

Figure 12: *Velocity (input) membership functions for the genetically tuned controller.*

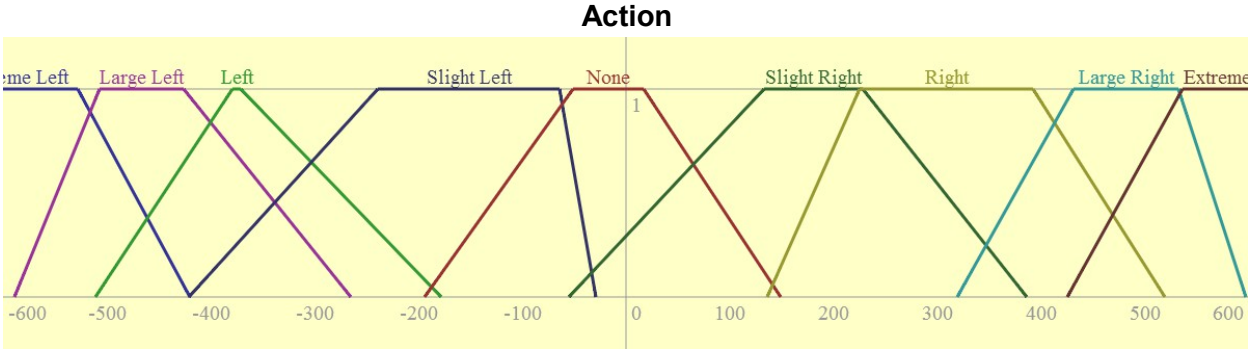

Figure 13: *Action (output) membership functions for the genetically tuned controller.*

These membership functions look a bit odd. Right away, one's attention is caught by the base points of the trapezoid in the position variable diagram (Figure 11). Almost all of them extend to

the center line. It seems that some behaviour where a bit of each function is involved became preferred in this genetic simulation. In all cases, larger extreme sections can be seen.

These changes in the membership functions are immediately apparent when one observes the controller in action. The car moves much more quickly to the line even when it is relatively close to it. The result does minimise the average distance from the line, but to this player's eye it seems a much less natural movement. It is still fairly smooth, however, and without the comparison to the manually tuned controller, it seems acceptably natural.

In terms of the mean and median distance, the genetically tuned controller does improve upon the manually tuned version. The mean drops to 25.75 and the median to 4.96. The histogram in Figure 14 below shows a similar distribution of distances to those from the manual controller.

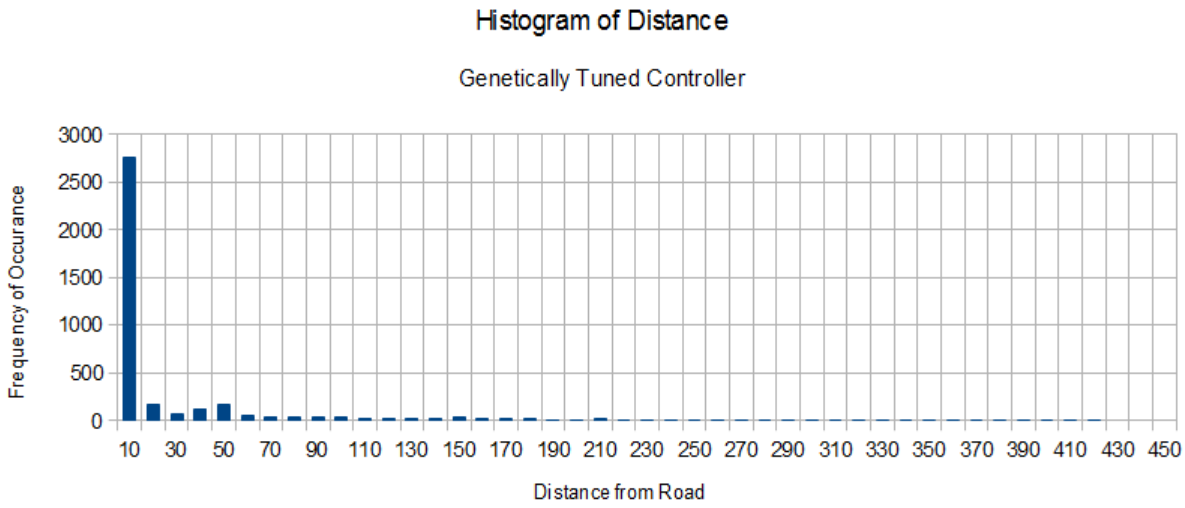

Figure 14: *Chart showing frequency of distances, in increments of ten, for the genetically tuned controller*

Track Effectiveness

The results presented above are only useful if the track succeeded in fully exercising the controller. This means making sure all membership functions and rules are executed. The pie charts in Figures 15-16 below show the amount of time the controller spent in each set during both the manual and genetic controller runs:

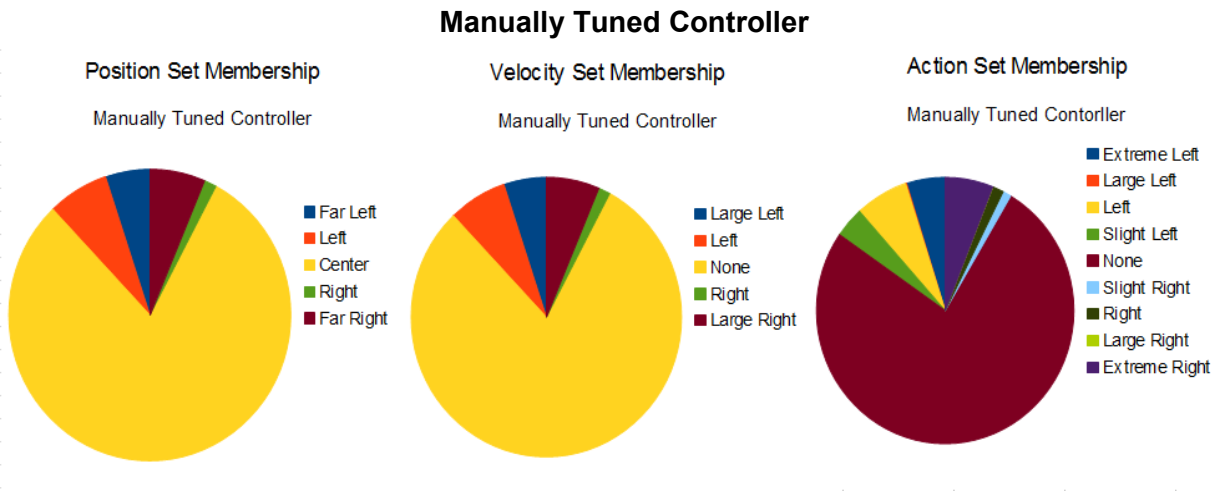

Figure 15: *Pie chart showing the percentage of time spent in each of the sets for the manually tuned controller*

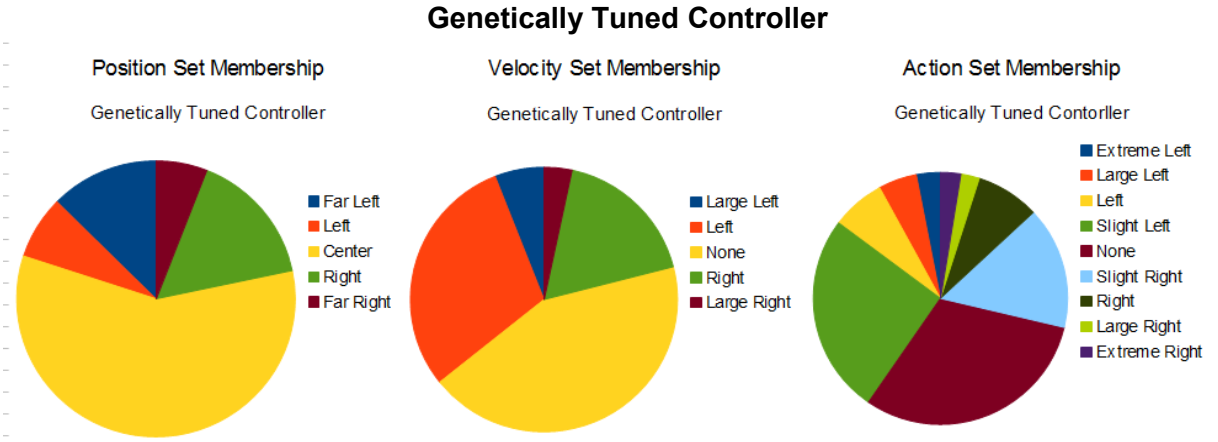

Figure 16: *Pie chart showing the percentage of time spent in each of the sets for the manually tuned controller*

It is clear that the track produced a more diverse response during the genetically tuned controller's run. However, in both runs, all sets are explored at least for a short time. Ultimately, it is difficult to design a track to work all areas of a controller equally within the confines of the proof of concept engine, without arbitrarily teleporting the road around or stopping and starting the simulation to set velocities and positions to specific points. A unit test outside the main engine may have been more appropriate for this purpose.

Conclusions

This project proved an effective vehicle for investigating the use of fuzzy logic controllers in racing games. Additionally, the effectiveness of using genetic algorithms to tune a fuzzy logic controller was explored. Ultimately, it seems that average distance from the line may not have

been a good measure of fitness for the genetic algorithm - while the genetic algorithm was very effective in creating a controller which minimised average distance from the line, the controller looked less “real” than the manually tuned version.

Given further opportunity for research, it would be interesting to explore other measures of fitness. For example, is it possible to quantify “smoothness”? Can this be incorporated into the genetic algorithm’s fitness measurement? Perhaps the genetic algorithm could be constrained to create only symmetric controllers. Alternatively, it would be interesting to perform more genetic optimisations and see what types of controllers emerge, then pull aspects of these generated controllers into a manually tuned version.

There are many possibilities, but it is clear from the project work that these methods have highly useful applications in game development.

References

Armstrong, J. (2009) Bezier Y at X Algorithm. [online] Available at: http://algorithmist.wordpress.com/2009/10/15/bezier-y-at-x-algorithm/ [Accessed: 12 Jan 2013].

BOURG, D. M., & SEEMANN, G. (2004). *AI for game developers*. Sebastopol CA, O'Reilly.

Buckler, C. (2011) How to Draw Bezier Curves on an HTML5 Canvas - SitePoint. [online] Available at: http://www.sitepoint.com/html5-canvas-draw-bezier-curves/ [Accessed: 12 Jan 2013].

Coolman, R. (2012) *Cubic Spline Functions*. Interviewed by Sarah Herzog [in person] 3 Jan 2013.

Hajek, P. (2010) Fuzzy Logic (Stanford Encyclopedia of Philosophy). [online] Available at: http://plato.stanford.edu/entries/logic-fuzzy/ [Accessed: 12 Jan 2013].

Holland, J. (n.d.) Genetic Algorithms - John H. Holland. [online] Available at: http://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm [Accessed: 12 Jan 2013].

Kamermans, M. (2012) Bézier curves - a primer. [online] Available at: http://processingjs.nihongoresources.com/bezierinfo/ [Accessed: 12 Jan 2013].

King, D. (2012) *CE1184A/K - Fuzzy Logic and Fuzzy State Machines*, Exeter: University of Abertay, Dundee.

National Instruments (2010) Defuzzification Methods (PID and Fuzzy Logic Toolkit) - LabVIEW 2010 PID and Fuzzy Logic Toolkit Help - National Instruments. [online] Available at:

http://zone.ni.com/reference/en-XX/help/370401G-01/lvpid/defuzzification_methods/ [Accessed: 12 Jan 2013].

Obitko, M. (1998) Genetic Algorithms. [online] Available at: http://www.obitko.com/tutorials/genetic-algorithms [Accessed: 12 Jan 2013].

Synaptic - The Peltarian Blog (2006) Fuzzy Math, Part 1, The Theory « Synaptic. [online] Available at: http://blog.peltarion.com/2006/10/25/fuzzy-math-part-1-the-theory/ [Accessed: 12 Jan 2013].

The University of Sheffield (2003) Cubic equations. [online] Available at: http://www.mash.dept.shef.ac.uk/Resources/web-cubicequations-john.pdf [Accessed: 12 Jan 2013].

Wolfram MathWorld (2013) Bézier Curve -- from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/BezierCurve.html [Accessed: 19 Dec 2012].

Wolfram MathWorld (2013) Cubic Formula -- from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/CubicFormula.html [Accessed: 12 Jan 2013].

Appendix A: Genetically Tuned Controller Export

Fuzzy Controller Data,
Position membership functions,
name,lbp,lpp,lc,rpp,rbp,rc,
Far Left,-650,-589,0,-519,-7,0,
Left,-523,-345,0,-269,-25,0,
Center,-230,-164,0,45,476,0,
Right,-7,75,0,177,577,0,
Far Right,35,378,0,506,650,0,
Velocity membership functions,
name,lbp,lpp,lc,rpp,rbp,rc,
Large Left,-650,-599,0,-496,-132,0,
Left,-496,-325,0,-162,10,0,
None,-282,-124,0,155,521,0,
Right,-77,131,0,275,585,0,
Large Right,142,210,0,650,650,0,
Action membership functions,
name,lbp,lpp,lc,rpp,rbp,rc,
Extreme Left,-650,-650,0,-528,-420,0,
Large Left,-589,-507,0,-426,-265,0,
Left,-511,-379,0,-371,-178,0,
Slight Left,-421,-239,0,-64,-29,0,

None,-194,-51,0,17,149,0,
Slight Right,-55,133,0,228,386,0,
Right,136,225,0,392,519,0,
Large Right,319,431,0,532,597,0,
Extreme Right,425,536,0,650,650,0,
Rules,
position,velocity,action,
0,0,0,
0,1,1,
0,2,2,
0,3,3,
0,4,4,
1,0,1,
1,1,2,
1,2,3,
1,3,4,
1,4,5,
2,0,2,
2,1,3,
2,2,4,
2,3,5,
2,4,6,
3,0,3,
3,1,4,
3,2,5,
3,3,6,
3,4,7,
4,0,4,
4,1,5,
4,2,6,
4,3,7,
4,4,8,