C++ Course Project Plan

qlearning2

Team Skele aihe

Esa Koskinen, Simo Muraja, Olli Kauppinen, Jussi Hirvonen
2016-11-12

Introduction and preliminary scope

Our aim with the project is to make a concise C++ application that can employ simple machine learning with the Q-learning technique. The basic idea is to have cars drive on a racetrack, and try to maximize the distance travelled in the shortest time whilst staying within the track limits. The application will have a simple GUI to observe the learning progress.

We also plan to include a “fast-forward n generations” -kind of feature in the application. If there is enough time, the scope might be extended into adding different types of cars or automatically generated tracks, etc. The general idea is to keep things easily configurable through tweaking some constants or a configuration file.

Basic architecture and main ASRs

The approach we took into designing the architecture was a very simple MVP-style one. Generally speaking, there is a separation of concerns between the UI side of things and how the nitty-gritty details of cars, tracks and the learning are modelled. It all comes together in a controller that acts on the user’s actions and takes care of updating the models.

Apart from the overarching architectural model, we have made a few decisions on how things should work.

First off, the representation model that will be used as each car’s “brains” will be based on simple neural networks. The exact details of how the ANN will be modelled and used has not been decided yet, but most likely we will be starting off with feedforward neural networks which use linear weighted sums and some kind of thresholds as a way of determining activation. The inputs could consist of a series of distances to walls in various directions. Other things that could be used as input for the network involve more details about the car’s state, as well as the currently cumulated reward and a bias unit. We will be exploring various alternatives to figure out what works; in the best case, the user can choose between different amounts of detail. If possible, we would like to use some kind of evolutionary algorithm to supplement Q-learning.

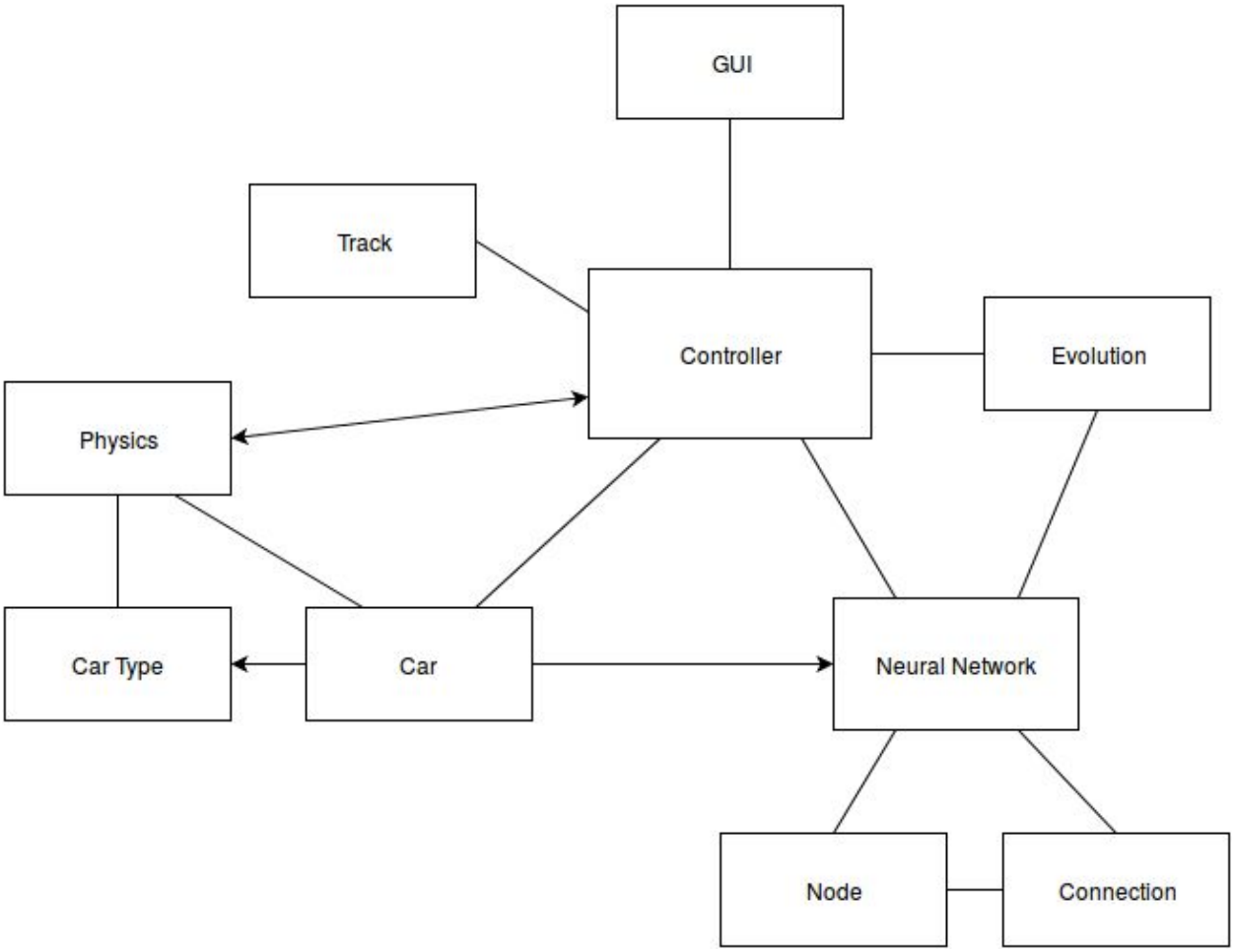
Secondly, each car instance represents a single car driving on a track; they have a type which determines the car’s parameters (acceleration, maximum speed, and so on) and a neural network which chooses actions for the car. The exact details of physics and the track are decoupled from the car’s representation and are not visible to the neural network, which should allow for easier division of development with less merge conflicts, not to mention making the learning environment more varied.

Thirdly, we have determined that as much learning history data should be saved on disk as is feasible. This has the advantage that if the cars' movement work in a deterministic manner rather than in a stochastic one, it should be possible to replay a specific car's run by just having access to its parameters and network. Apart from that, we think that it would be nice to also store and retrieve tracks and other such things using files.

All in all, the aim is to keep program structure simple and make individual classes fairly small with well-defined responsibilities and interfaces. Configurability is important for us. The UI side is purposefully left very open-ended at this point in time, as we want to have a functioning logic side before using a lot of time on finding the best representation of the learning process.

Preliminary class division

What follows is a rough sketch on how some of the classes could be divided and what kind of information exchange there is between them. Each class has a short description below.



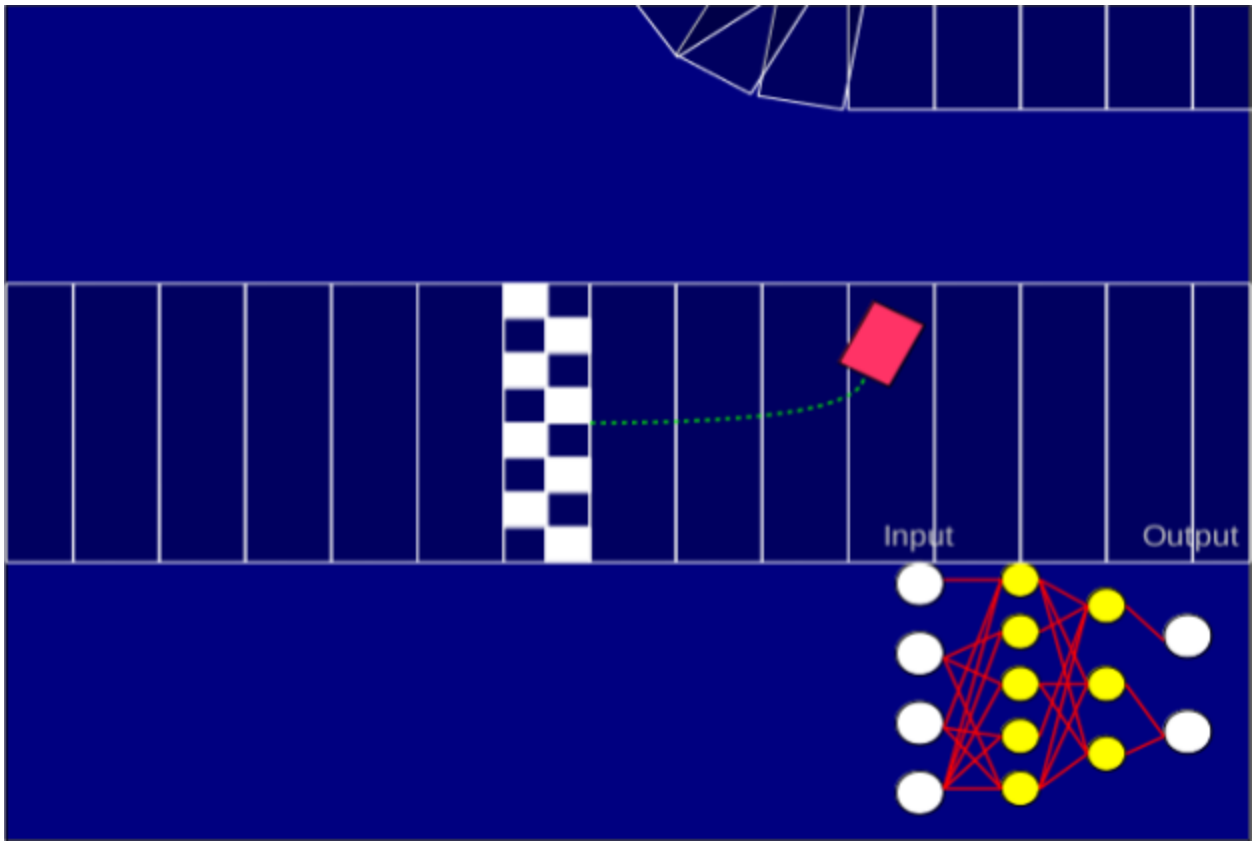
Dia 1: A very rough division of classes

Controller

Running the program creates a single Controller instance, which keeps the whole thing together. It handles user requests, runs cars through the track, figures out their reward values and calls Evolver on their NN's accordingly, keeps track of things and handles possible file IO. The controller is initialized with various constants which determine the exact modes of working for the other parts of the software.

GUI

As instructed in the assignment, we are going to build a basic graphical user interface to illustrate the functionality of our application. The GUI obviously includes representations of cars and the track. In addition, we plan on displaying a visualization of the neural network to the user. The GUI will initially showcase the car with the highest fitness in the generation. However, it will be possible to replay the execution of any given car. Other actions that the user could do include moving forward and backward between generations, compute a large number of generations at once and modify the physics of the system and/or the amount of cars per generation. We remain undecided about whether the user commands should have buttons and menus in the GUI. An alternative would be to issue commands from command line and have the GUI as its own separate thing.



Dia 2: A vision of GUI

Track

We have come up with several ways to incrementally develop the racetrack's design. In the first two weeks, we are going to build a cut down version where straying from the track immediately disqualifies the ongoing simulation. The track could consist of small pieces that are laid out successively. The car could—for example—receive a reward inversely proportional to the time it took to get to a new section of the track. We thought about allowing the car to race outside the track to see if it would find shortcuts, but decided against this because removing the track limits as a constraint would make it difficult to decide when to stop the simulation—there would be no way to know if the car was actually making progress. If we have time, we want to implement a random track generation algorithm that would allow us to change the racetrack between generations.

Car and CarType

A Car is basically a container for the car's state as it runs through the track. It has knowledge of its position, direction, and other such parameters. The CarType of the car determines what kind of effect each of the car's actions has; in practice, it holds information about the car's acceleration, turning rate, maximum speed, etc. We separated these two in order to later test the difference between more "general purpose" networks which have to handle all types of cars, and specialized networks which have only been trained on a single car's parameters.

Physics

When the Car's neural network decides on an action, the Car then asks the Physics class to adjust its state appropriately. This separation of the movement's details into its own class should allow us to easily implement more complex physics engines later on in development with less conflicts. We are planning to start off with our own very simple physics system, but are interested in implementing Box2D or similar at some point.

Neural Network, Node, Connection

The NN-class is a data structure which holds Node objects and takes care of figuring out an action based on some set of inputs. We are currently undecided on whether connections between nodes should be separated into its own class or not; there is a clear trade-off with using development time in order to have a framework that can support a more complex node system. The Node class may need further dividing into input, hidden and output nodes.

Evolution

Evolution is a helper class for Controller with all of the needed functions to create, remove, alter, evolve and manage the neural networks. The separation is done to clarify code and to prevent the Controller from being bloated too much. After all, most of these functions are not needed anywhere else, and it helps with focusing all Q-learning related functionality into the same place.

Preliminary schedule and responsibilities

Our first aim is to have a working prototype ready after two weeks. This prototype should have a rudimentary GUI and the necessary logic implemented for some basic learning.
Simo and Esa are responsible for the implementation of the Q-learning algorithm and neural networks. Olli will take care of our instrument for demonstrating the Q-learning process, namely the car and physics related to it. Jussi is in charge of creating the user interface and the track on which the car runs.

Working practices

Git

Our git will have a master branch, a release candidate branch and for each feature a rough development branch. Release candidate will be pushed to master once ready. Feature branches are supposed to pass tests and be reviewed by at least one person besides the responsible for that feature before merging into release candidate. Individual team members have their own dev branches for specific feature until pushing and merging to the feature development branch.

Meetings and working

We’re planning to meet weekly for a review, planning and a coding session. The rest of the work will be done individually. We have estimated that each group member will have to spend roughly one day per week programming outside of the group meetings in order for us to complete the project in time.

Communication

A Telegram group that we created for the project is already in good use, and the team has agreed to mostly use it for communication. Slack will probably only be used if we run into a problem that we can’t solve.