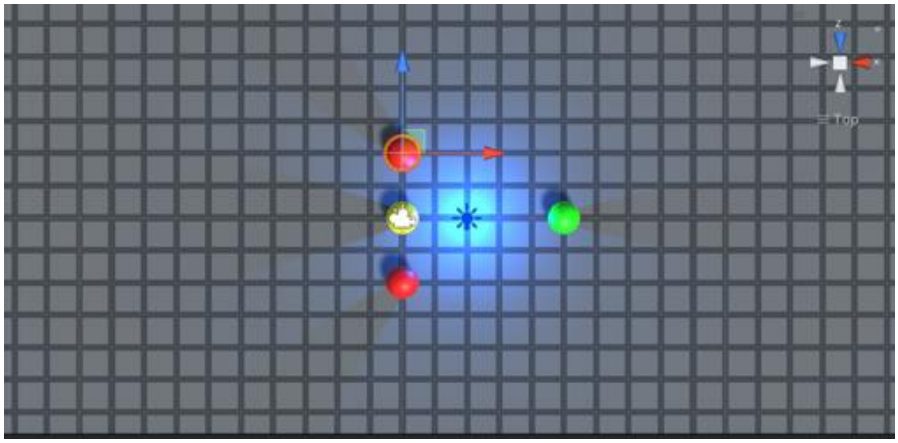
Game



As a part of the project, the game played an extremely important role in collecting experimental data, which will later be used to train the models.

There are two important reasons to use games to collect experimental data:

1. Compared to data collected from real life or existing data sets on the web, we have more freedom to obtain data in different dimensions during the experiment.

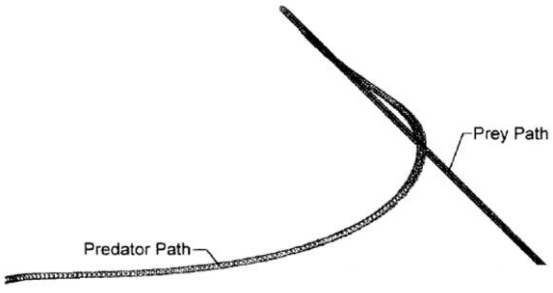
2. The game environment is ideal and not subject to accidental interference, so it is of low noise. This is more friendly to us who are new to neural networks.

Besides it is also interesting to create our own unique dataset.

Goal of the player:

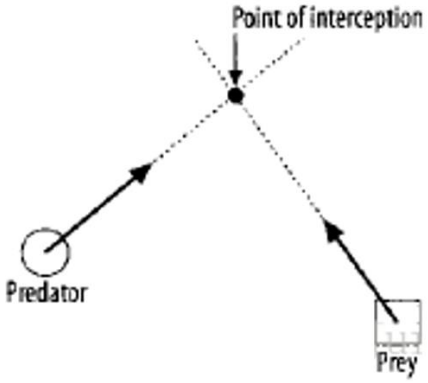
During the game, the player controls the yellow ball and tries to kick the green ball into the light spot, and he should try to Prevent the yellow ball from hitting the two red balls who are the enemy. When the green ball and the light point collide, the light point will appear at the next random location. The player will repeat this process for 15 minutes

The strategy of Enemy1:



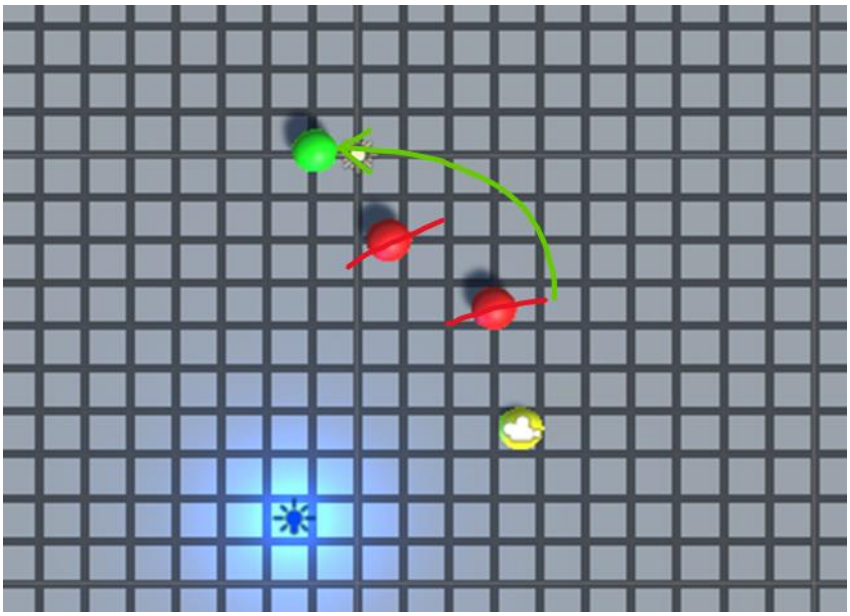
Enemy1's strategy is eye tracking, which means its speed direction is always toward the player. In the picture above the predator performs like a tracking missile which is exactly what enemy1 do to the yellow ball.

The strategy of Enemy2:



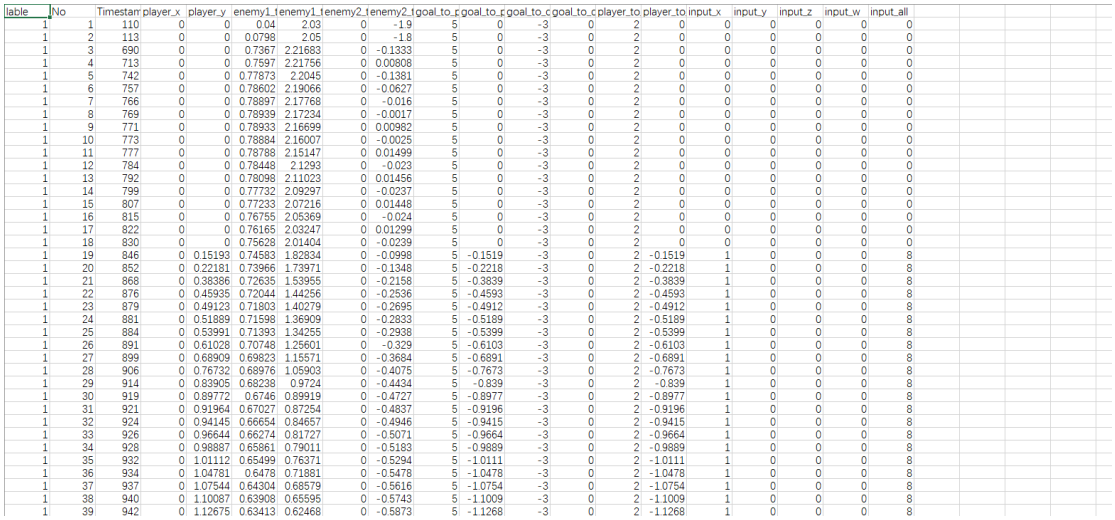
Enemy2's strategy is a little different from that of Enemy1, it has the ability to predict--- it will read the player's velocity vector. It's more about intercepting players instead of just chasing

The green ball's movement:



For green ball, it will flee away if the player is too closed to it. To increase the playability and difficulty of the game, a program is written to make the green ball tend to hide behind the two red balls, which means it will use the two red balls to flee away from the player, such behavior is shown in the picture above.

The recording of data:



lable	No	Timestam	player_x	player_y	enemy1_t	enemy1_t	enemy2_t	enemy2_t	goal_to_p	goal_to_p	goal_to_c	goal_to_c	player_to	player_to	input_x	input_y	input_z	input_w	input_all
1	1	110	0	0	0.04	2.03	0	-1.9	5	0	-3	0	2	0	0	0	0	0	0
1	2	113	0	0	0.0798	2.05	0	-1.8	5	0	-3	0	2	0	0	0	0	0	0
1	3	690	0	0	0.7367	2.21683	0	-0.1333	5	0	-3	0	2	0	0	0	0	0	0
1	4	713	0	0	0.7597	2.21756	0	0.00808	5	0	-3	0	2	0	0	0	0	0	0
1	5	742	0	0	0.77873	2.2045	0	-0.1381	5	0	-3	0	2	0	0	0	0	0	0
1	6	757	0	0	0.78602	2.19066	0	-0.0627	5	0	-3	0	2	0	0	0	0	0	0
1	7	766	0	0	0.78897	2.17768	0	-0.016	5	0	-3	0	2	0	0	0	0	0	0
1	8	769	0	0	0.78939	2.17234	0	-0.0017	5	0	-3	0	2	0	0	0	0	0	0
1	9	771	0	0	0.78933	2.16699	0	0.00982	5	0	-3	0	2	0	0	0	0	0	0
1	10	773	0	0	0.78884	2.16007	0	-0.0025	5	0	-3	0	2	0	0	0	0	0	0
1	11	777	0	0	0.78788	2.15147	0	0.01499	5	0	-3	0	2	0	0	0	0	0	0
1	12	784	0	0	0.78448	2.1293	0	-0.023	5	0	-3	0	2	0	0	0	0	0	0
1	13	792	0	0	0.78098	2.11023	0	0.01456	5	0	-3	0	2	0	0	0	0	0	0
1	14	799	0	0	0.77732	2.09297	0	-0.0237	5	0	-3	0	2	0	0	0	0	0	0
1	15	807	0	0	0.77233	2.07216	0	0.01448	5	0	-3	0	2	0	0	0	0	0	0
1	16	815	0	0	0.76755	2.05369	0	-0.024	5	0	-3	0	2	0	0	0	0	0	0
1	17	822	0	0	0.76165	2.03247	0	0.01299	5	0	-3	0	2	0	0	0	0	0	0
1	18	830	0	0	0.75628	2.01404	0	-0.0239	5	0	-3	0	2	0	0	0	0	0	0
1	19	846	0	0.15193	0.74583	1.82834	0	-0.0998	5	-0.1519	-3	0	2	-0.1519	1	0	0	0	8
1	20	852	0	0.22181	0.73966	1.73971	0	-0.1348	5	-0.2218	-3	0	2	-0.2218	1	0	0	0	8
1	21	868	0	0.38386	0.72635	1.53955	0	-0.2158	5	-0.3839	-3	0	2	-0.3839	1	0	0	0	8
1	22	876	0	0.45935	0.72044	1.44256	0	-0.2536	5	-0.4593	-3	0	2	-0.4593	1	0	0	0	8
1	23	879	0	0.49123	0.71803	1.40279	0	-0.2695	5	-0.4912	-3	0	2	-0.4912	1	0	0	0	8
1	24	881	0	0.51889	0.71598	1.36909	0	-0.2833	5	-0.5189	-3	0	2	-0.5189	1	0	0	0	8
1	25	884	0	0.53991	0.71393	1.34255	0	-0.2938	5	-0.5399	-3	0	2	-0.5399	1	0	0	0	8
1	26	891	0	0.61028	0.70748	1.25601	0	-0.329	5	-0.6103	-3	0	2	-0.6103	1	0	0	0	8
1	27	899	0	0.68909	0.69823	1.15571	0	-0.3684	5	-0.6891	-3	0	2	-0.6891	1	0	0	0	8
1	28	906	0	0.76732	0.68976	1.05903	0	-0.4075	5	-0.7673	-3	0	2	-0.7673	1	0	0	0	8
1	29	914	0	0.83905	0.68238	0.9724	0	-0.4434	5	-0.839	-3	0	2	-0.839	1	0	0	0	8
1	30	919	0	0.89772	0.6746	0.89919	0	-0.4727	5	-0.8977	-3	0	2	-0.8977	1	0	0	0	8
1	31	921	0	0.91964	0.67027	0.87254	0	-0.4837	5	-0.9196	-3	0	2	-0.9196	1	0	0	0	8
1	32	924	0	0.94145	0.66654	0.84657	0	-0.4946	5	-0.9415	-3	0	2	-0.9415	1	0	0	0	8
1	33	926	0	0.96644	0.66274	0.81727	0	-0.5071	5	-0.9664	-3	0	2	-0.9664	1	0	0	0	8
1	34	928	0	0.98887	0.65861	0.79011	0	-0.5183	5	-0.9889	-3	0	2	-0.9889	1	0	0	0	8
1	35	932	0	1.01112	0.65499	0.76371	0	-0.5294	5	-1.0111	-3	0	2	-1.0111	1	0	0	0	8
1	36	934	0	1.04781	0.6478	0.71881	0	-0.5478	5	-1.0478	-3	0	2	-1.0478	1	0	0	0	8
1	37	937	0	1.07544	0.64304	0.68579	0	-0.5616	5	-1.0754	-3	0	2	-1.0754	1	0	0	0	8
1	38	940	0	1.10087	0.63908	0.65595	0	-0.5743	5	-1.1009	-3	0	2	-1.1009	1	0	0	0	8
1	39	942	0	1.12675	0.63413	0.62468	0	-0.5873	5	-1.1268	-3	0	2	-1.1268	1	0	0	0	8

The data is recorded in real time in a csv file. As shown in the figure above, the first column shows the classification label, the second column shows the frame number, and the third column shows the timestamp. Starting from the fourth column, a combination of every two columns describes the relative position relationship between objects in the form of vectors. The last five columns show the player's input in binary and hexadecimal format.

LSTM Model for analysis

Data preprocessing

Data preprocessing consists mainly of the following steps:

1. deviation calculation:
   When processing the vector data and the timestamp, the entire table is moved down one row and subtracted from the previous table, resulting in a new table. The data in the new tables are much smaller because they represent changes in the position of objects over milliseconds. As shown in the picture below:

G	H		G	H
enemy1_tc	enemy1_tc	1	enemy1_tc	enemy1_tc
0.04	2.03	2	0.04	2.03
0.079803	2.050002	3	0.039803	0.020002
0.40337	1.883494	4	0.6569	0.166825
0.436713	2.223702	5	0.022994	0.000735
0.446782	2.332035	6	0.019028	-0.01307
0.449318	2.361158	7	0.007299	-0.01384
0.450807	2.379655	8	0.002948	-0.01298
0.451678	2.397227	9	0.000417	-0.00534
0.452402	2.415191	10	-5.86E-05	-0.00536
0.452767	2.433425	11	-0.00049	-0.00692
0.453104	2.471732	12	-0.00097	-0.0086
0.453018	2.504459	13	-0.0034	-0.02217
0.452759	2.529243	14	-0.00349	-0.01907
0.452284	2.549736	15	-0.00366	-0.01726
0.451647	2.575002	16	-0.00499	-0.02081
0.450726	2.598595	17	-0.00479	-0.01847
0.449984	2.618988	18	-0.0059	-0.02122
0.449082	2.645139	19	-0.00537	-0.01843
0.448095	2.665586	20	-0.01045	-0.1857
0.447273	2.684766	21	-0.00617	-0.08863
0.44633	2.701197	22	-0.01331	-0.20016
0.445536	2.717377	23	-0.00591	-0.09699
0.444825	2.733791	24	-0.00242	-0.03977
0.443999	2.748173	25	-0.00205	-0.0337
0.443191	2.764668	26	-0.00205	-0.02654
0.442334	2.784512	27	-0.00644	-0.08654
0.441354	2.801777	28	-0.00925	-0.1003
0.440446	2.820519	29	-0.00848	-0.09668
0.4395	2.835616	30	-0.00737	-0.08663

The normalization is also done by the way.

2. detect interfering rows of data:
   While the game is an ideal environment for experimentation, there are still some exceptions. Take the simplest example, as shown below:

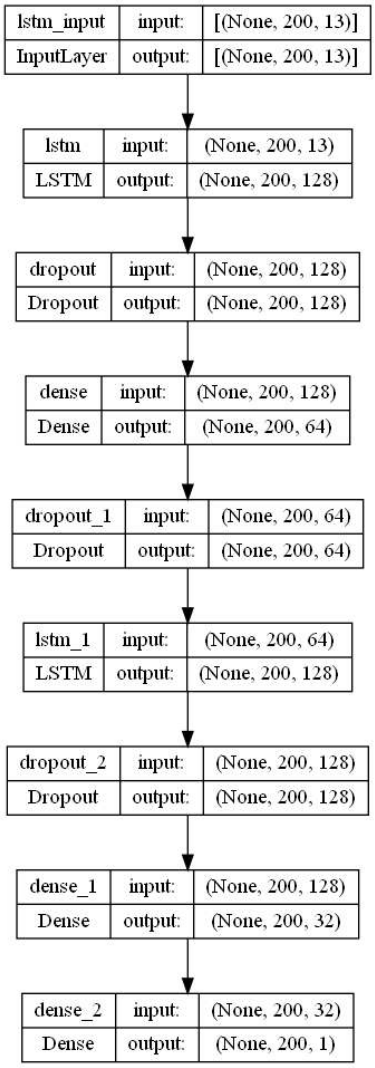
Timestamp
110.0
3.0
577.0
23.0
29.0
15.0
9.0
3.0
2.0
2.0
4.0
7.0

When you take a closer look at 'Timestamp' column after taking the deviation, you'll notice that some frames take significantly longer to execute than others, and this breaks the consistency of the data recording, which can have a huge impact on subsequent training. It is important to mark them and avoid using them in the LSTM training.

3. select proper columns:
   Data in different columns may represent the same information, and it will be inefficient to bring them all into the sequence. Therefore, in the training of LSTM, I removed all the input data from the user, because their coordinates and the player's coordinates represented repeated information

Model structure

The structure of the model is shown in the following figure. In the later part of parameter adjustment, I adjusted the length and dimension of input data and the number of neurons in the model, but the overall framework did not change:



A total of two LSTMS and three dense layers are used in this model. The last dense layer's activation function is 'sigmoid'.

The loss function, optimizer and metrics are shown as below:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['binary_accuracy'])
```

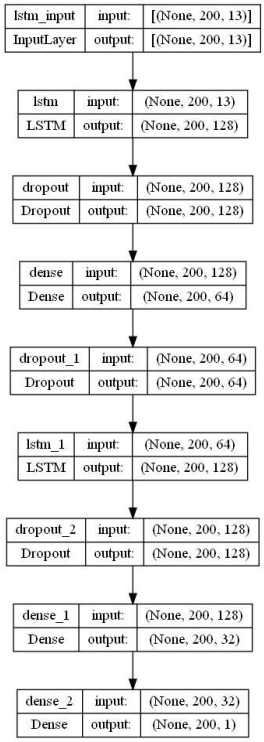
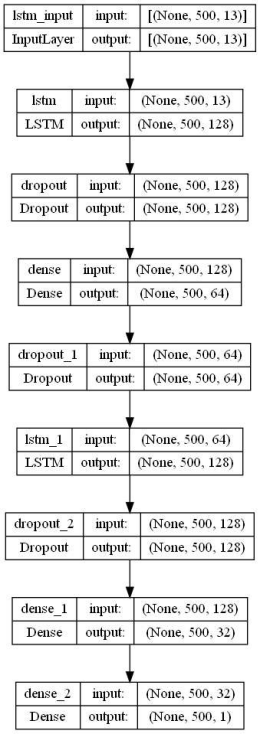
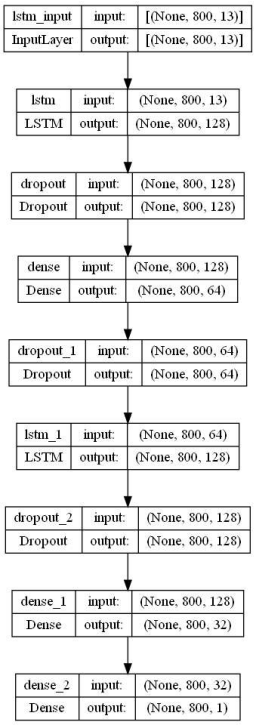
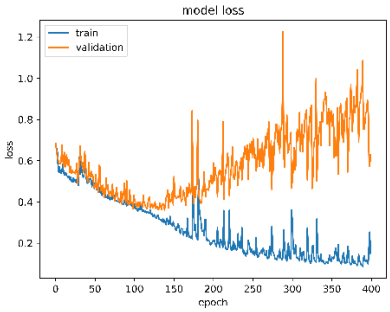
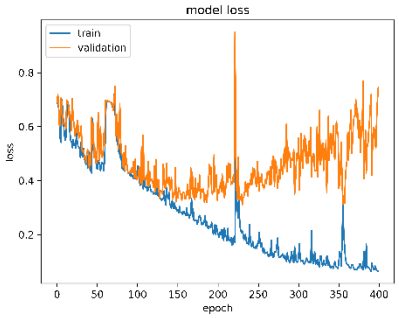
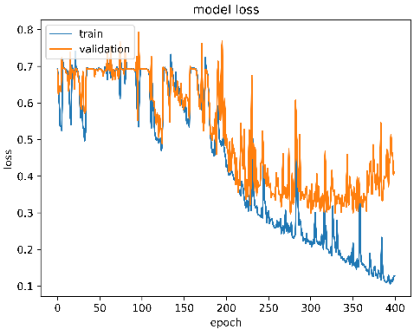
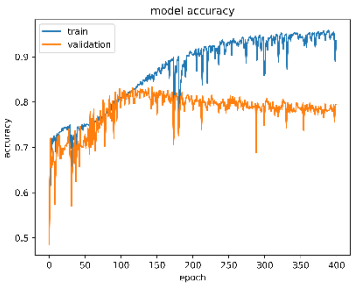
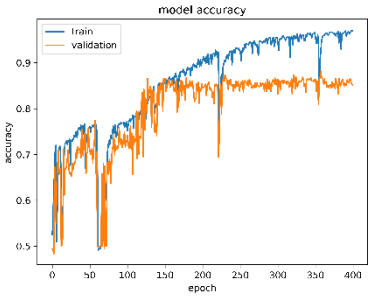
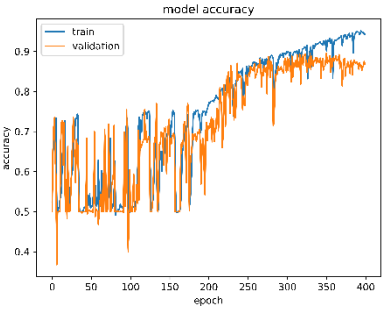
Result analysis and parameter tuning

Thanks to cudNN by Nvidia, Training tasks can be completed in less time

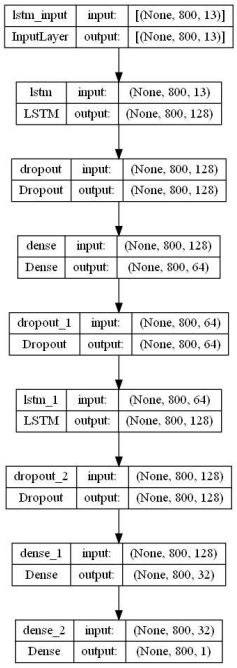
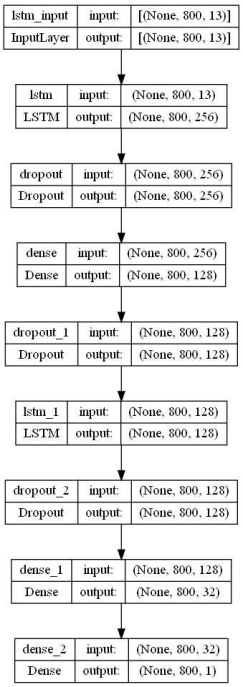
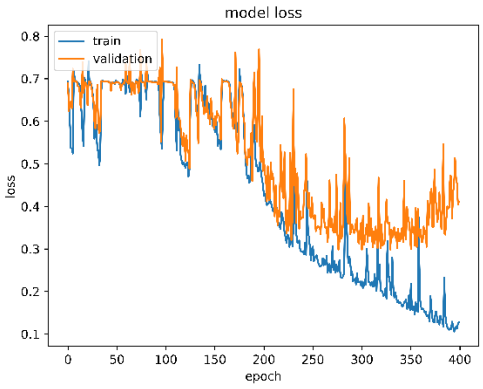
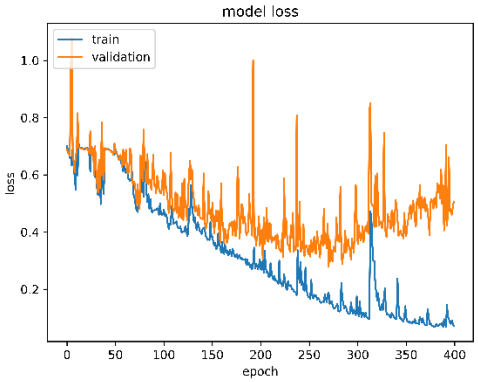
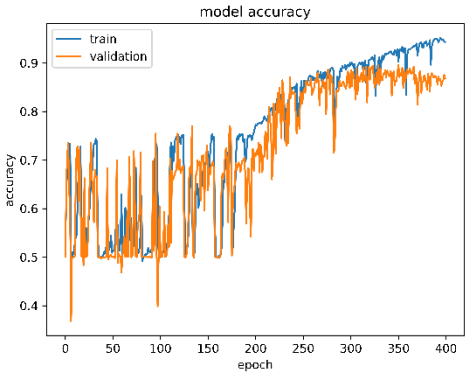
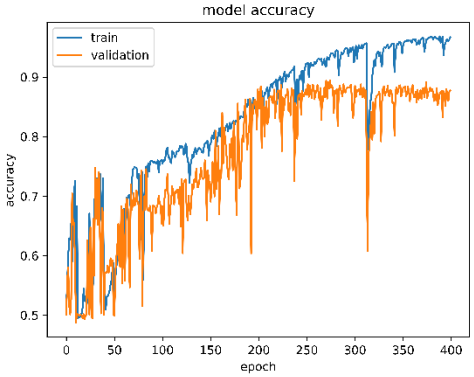
In the parameter adjustment part, the learning rate is 0.001 by default, and the batch size is not adjusted either, it is set to 32.

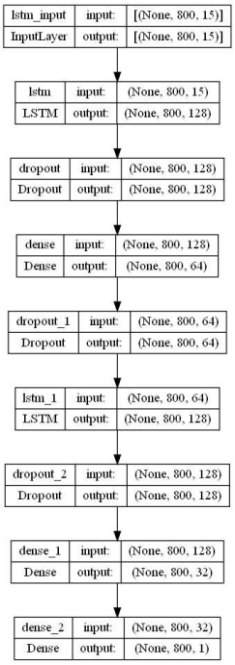
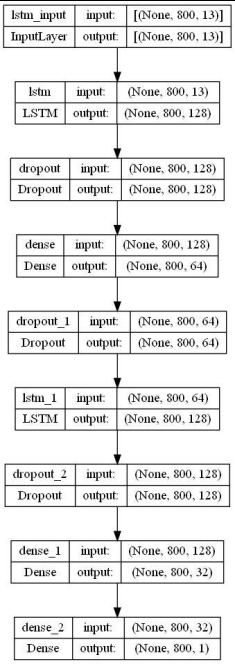
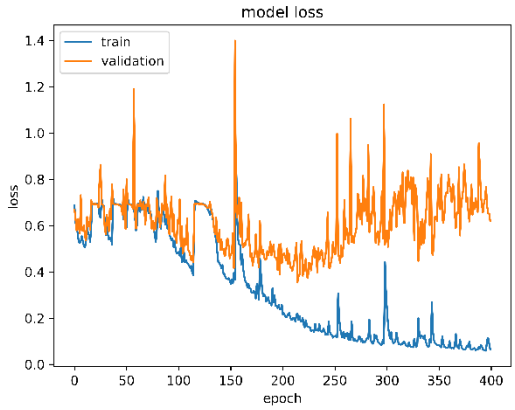
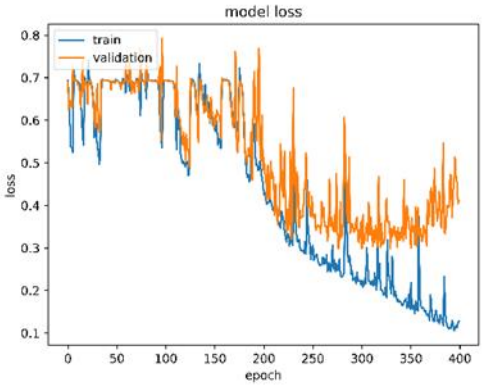
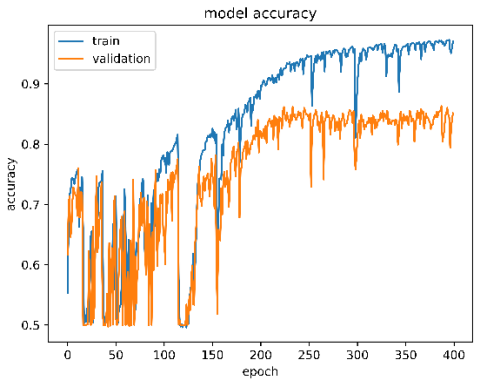
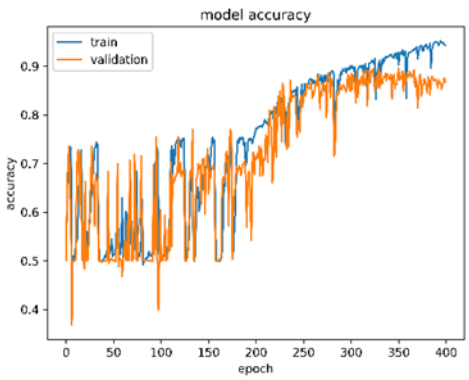
The emphasis of parameter adjustment is on length adjustment of input time series and neuron number adjustment.

Here are a few comparisons:

Input shape	(200,13)	(500,13)	(800,13)
Model structure			
Loss			
accuracy			

Comments: According to the loss values, different degrees of overfitting occurred in the later stage of the training of the three models. With the increase of the input data time span, the overfitting time also appeared later. Longer time series also take longer to train, and the accuracy of the validation set increases with the increase of step size. Extrapolating from these three sets of data, 500 to 800 is the optimal time series length.

Neuron number	-less	+more
Model structure		
Loss		
accuracy		
Comments: Different degrees of overfitting occurred in the later stage of the training, There was no significant difference		

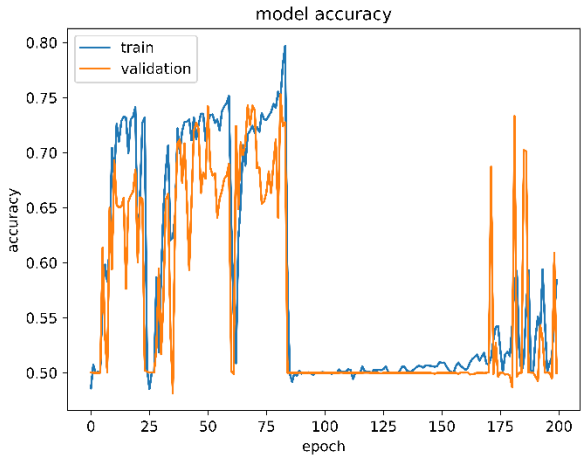
Different inputs	Keyboard inputs	Player's coordinates
Model structure		
Loss		
accuracy		
Comments: When a set of continuous data (Player's coordinates) is replaced by discrete data (Keyboard inputs), the training effect becomes worse, overfitting occurs earlier and the accuracy of validation set decreases.		

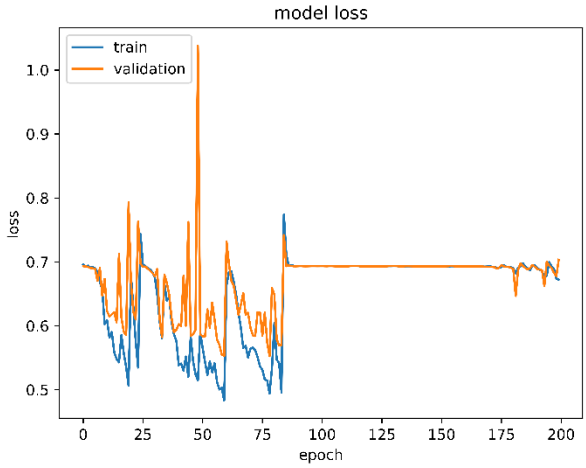
The validation set accuracy is mostly maintained at 85% after training, which is satisfactory.

Interesting phenomenon during the optimization

1. Whether or not to include a timestamp?

In the first few sessions, I didn't include timestamp information considering it as superfluous, and I got very bad results. As shown in the picture below:





The loss function diverges randomly at first, and then completely stabilizes around 0.7. My guess is that it gets stuck in a local optimum. No matter how I adjust the parameters, the end result is the same, without exception.

Due to hardware uncertainty, the refresh interval between frames is not constant, and it is really hard for the neuron network to figure it out, which may need a much more complex network structure.

It may seem unimportant, but Omit timestamp input will significantly do damage to the fluency and consistency of the time series, which may cause LSTM a lot of trouble.

2. Vectors or polar coordinates?

At first, I wanted to convert the vectors to polar coordinates because I thought it would better show the player's sense of control over distance from enemies or the goal, which may allow neural networks to learn faster and better. But the results were far from ideal. The accuracy of the validation set has been fluctuating around 50%.

G	H		G	H
enemy1_tc	enemy1_tc	1	enemy1_tc	enemy1_tc
0.04	2.03	2	1.551094	2.030394
0.079803	2.050002	3	1.531888	2.051555
0.736703	2.216827	4	1.249955	2.336034
0.759697	2.217562	5	1.240745	2.344082
0.778726	2.204495	6	1.231234	2.337993
0.786025	2.190657	7	1.226297	2.327405
0.788972	2.177681	8	1.223207	2.316198
0.789389	2.172344	9	1.222251	2.311323
0.789331	2.166988	10	1.221481	2.30627
0.788845	2.160069	11	1.22065	2.299603
0.787876	2.151471	12	1.21976	2.291195
0.784476	2.129304	13	1.217808	2.269215
0.780983	2.11023	14	1.216334	2.250112
0.777318	2.092967	15	1.21519	2.232652
0.772333	2.072159	16	1.214027	2.211411
0.767547	2.053692	17	1.213131	2.192437
0.761652	2.032472	18	1.212252	2.170497
0.756278	2.014037	19	1.211584	2.151349

I realized later that when I converted the vector to polar coordinates, after the object moved about a week relative to the reference, the Angle in polar coordinates would change dramatically, which changes from -3.14 to 3.14 or from 3.14 to -3.14. Such sudden changes in values require more complex network structures and more neurons to learn, So I eventually gave up on using polar coordinates instead of vectors.

-8.25777	15100	-3.13586
-8.2495	15101	-3.13922
-8.24155	15102	3.140275
-8.23644	15103	3.13762
-8.22928	15104	3.134549

A study of player behavior patterns towards high-threat objects based on eye movement and behavioral analysis

The goal of the project

The goal of the project is to figure out the differences in the player's behavior patterns in the face of different threatening objects in the game, and to use these patterns to backtrack to figure out which objects are perceived as high threat by the player.

The main body of this project is a contract test. The experimental environment is the game mentioned above. The input data is also processed using the neural network mentioned above. Therefore, all the above can be regarded as a preliminary experiment, through which it has been proved that relevant games and data analysis methods can be applied to this project.

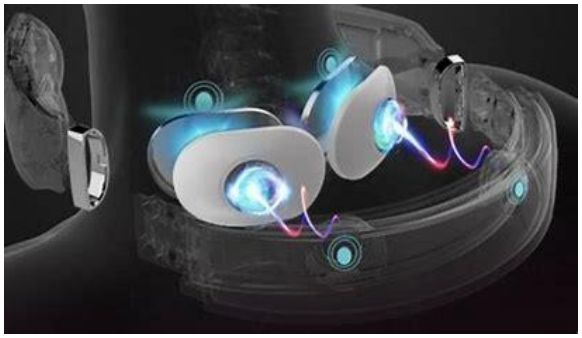
new elements compared to the preliminary experiment:

eye movement:

It will be conducted using VR equipment, for that the VR devices can have the highest accuracy. And the AOI method will be applied in this project to study the users' area of interest, thus analyze where the user is looking at.

Small electric shock device:

It will provide a feel of acupuncture towards the players as a punishment if they fail to keep away from the red balls (the enemies).



Experiment

Each participant will take part in one of two controlled experiments. In the first part, the player is not punished for touching the red ball except that the screen turns red. In the second

part, a small electric shock device will give the player a small stimulus to increase the threat of the red ball.

In the experiment, two sets of data are collected, one is the relative change in the position of the ball generated by user input, and the other is the user's eye movement data. The former will be processed by the neural network according to the pre-experimental method. Eye movement data will be processed by traditional methods together with neural network.