**Python项目报告**

**深度强化学习工具箱**

**DRL Tool Box**

**学院：岭南学院**

**组员：梁智鹏、陈昊、张意伟**

**学号：16331021、16332008、16332223**

**完成日期：2018.6.25**

**指导老师：张宏斌**

**目录**

[文件树： 4](#_Toc518086297)

[配置情况： 5](#_Toc518086298)

[一、 研究意义 6](#_Toc518086299)

[1.1 强化学习的缘起 6](#_Toc518086300)

[1.2 强化学习运用的延伸 6](#_Toc518086301)

[1.3 项目目的 6](#_Toc518086302)

[二、 理论基础 7](#_Toc518086303)

[2.1强化学习是什么？ 7](#_Toc518086304)

[2.2 MDP马尔科夫决策过程 7](#_Toc518086305)

[2.3 Bellman方程 8](#_Toc518086306)

[2.4 优化估值函数：最优动作估值函数 8](#_Toc518086307)

[2.5 优化迭代方法：Q learning 9](#_Toc518086308)

[2.6 如何生成动作？ 9](#_Toc518086309)

[2.7 深度学习神经网络 9](#_Toc518086310)

[2.8 神经网络算法讲解： 10](#_Toc518086311)

[2.8.1 DQL.py： 10](#_Toc518086312)

[2.8.2 DQLBrain算法详解： 11](#_Toc518086313)

[2.8.3 训练过程： 12](#_Toc518086314)

[三、 测试小游戏 14](#_Toc518086315)

[3.1 跳跳人 14](#_Toc518086318)

[3.1.1 涉及模块 14](#_Toc518086319)

[3.1.2 游戏规则 14](#_Toc518086320)

[3.1.3 游戏画面 14](#_Toc518086321)

[3.1.4 模块介绍 14](#_Toc518086322)

[3.1.5 设计思路 15](#_Toc518086323)

[3.2 贪吃蛇 16](#_Toc518086324)

[3.2.1 涉及模块 16](#_Toc518086325)

[3.2.2 游戏规则 16](#_Toc518086326)

[3.2.3 游戏画面 16](#_Toc518086327)

[3.2.4 设计思路 16](#_Toc518086328)

[3.2.5 游戏流程图 17](#_Toc518086329)

[四、 工具箱窗口设计 17](#_Toc518086330)

[4.1涉及模块： 17](#_Toc518086331)

[4.2 设计思路 17](#_Toc518086332)

[4.2.1 示意图 17](#_Toc518086333)

[4.2.2 主窗口功能详解 18](#_Toc518086334)

[4.2.3 主窗口设计思路 24](#_Toc518086335)

[4.2.4 设置参数菜单设计思路： 24](#_Toc518086336)

[4.2.4 与数据库的交互 25](#_Toc518086337)

[4.2.5 服务器设计思路： 25](#_Toc518086338)

[五、 项目困难及解决方案 26](#_Toc518086339)

[六、 项目应用： 27](#_Toc518086340)

[七、 项目分工 29](#_Toc518086341)

[项目时间进度表： 29](#_Toc518086342)

[项目分工： 29](#_Toc518086343)

[备注： 29](#_Toc518086344)

[八、 参考文献 30](#_Toc518086345)

# 文件树：

**RLToolBox:**

|————**MyLibrary.py** 用于设置游戏中人物等类

|————**run\_window.py** 启动主程序，包括启动界面

|————**mainwindow.py** 主界面程序

|————**setting.py** 参数调节窗口程序

|————**message\_box.py** 消息框窗口程序

|————**DQL.py** 人工智能主程序，负责选择和启动游戏、启动深度强化学习内核

|————**DQLBrain.py** 深度强化学习内核

|————**game\_setting.py** 存储已有游戏决策状态数、库名等信息，新游戏加入必须将相关信息也加入在其中

|————**flask\_tk.py** 服务器文件

|————**jumpMan.py** 跳跳人游戏文件

|————**greedySnake.py** 贪吃蛇游戏文件

|————**resource** 窗口图片资源文件夹

|————**save\_networks** 已得出的模型文件

|————**templates**

**|————index**.**html** 网页前端模板文件

|————**static**

|————**exporting**.**js**

|————**highcharts-zh\_CN.js**

|————**highstock.js**

|————**jquery.js**

|————**temp**.**db** 临时数据库，用于服务器和AI端数据交互使用

|————**greedy**\_**snake**.**data**-**00000**-**of**-**00001**

|————**greedy**\_**snake**.**index**

|————**greedy**\_**snake**.**meta** 以上三个为一个训练好的模型

|————**greedy**\_**snake**.**db**.**bak**

|————**greedy**\_**snake**.**db**.**dat**

|————**greedy**\_**snake**.**db**.**dir** 以上三个为一个模型文件

|————**setting**\_**resource**.**py** 设定窗口的资源文件

|————**resource**\_**message**\_**box**.**py** 消息框窗口的资源文件

|————**resource**.**py** 主窗口的资源文件

|————**document**.**py** 根据数据库文件自动化生成报告

# 配置情况：

* **Python 3**
* **tensorflow-gpu**
* **pygame**
* **opencv-python**
* **PyQt5**
* **sys**
* **Threading**
* **Multiprocessing**
* **Shelve**
* **os**
* **sqlite3**
* **socket**
* **pyperclip**
* **flask**
* **glob**
* **shutil**
* **numpy**
* **pandas**
* **time**
* **importlib**

# 研究意义

### 1.1 强化学习的缘起

心理学界早在二十世纪已经提出一种行为学习的模式——强化学习，其主要观点认为行为的形成，是由于行为后果导致的。即如果一个行为能够带来正向的后果，那么该后果会增大同样场景之下该行为出现的概率，反之亦然。

后果

行为

而近年来，强化学习成为了人工智能研究的一个热点。强化学习由于能够无视环境和具体规则的约束，真正从空白通过不断学习，最终掌握极其复杂甚至超出人类想象的策略而广受关注。而随着深度学习算法和框架迅猛发展和越加成熟，深度学习与强化学习的结合更是极大地增强了后者的表现。2013年DeepMind公司在NIPS上发布了Playing Atari with Deep Reinforcement Learning 一文，首次将深度学习和强化学习结合起来，并取得了超出已有强化学习框架的表现。而在之后AlphaGo和AlphaZero战胜了人类之后，强化学习更是受到了极大的关注。

### 1.2 强化学习运用的延伸

强化学习除了在游戏AI中运用之外，在**金融**、**物流调度**方面当中也开始崭露头角。早在2002年，Ilaria等人就开始了强化学习在**库存管理**方面的研究。而在金融方面，2017年Zhengyao Jiang等人尝试了深度强化学习在**资产配置**中的应用，并展现出其不凡的表现。

无论是深度学习还是强化学习，超参数的选择都是决定模型表现至关重要的因素，但是涉及到的超参数往往不少，而且目前还缺乏系统科学的超参数调节方法。另外在强化学习中，机制例如激励机制的设置对于最后学习的效果也是极其关键的。例如，当容错率过低的时候，算法由于对状态空间的学习不足，最终表现往往不佳；激励机制如果设置过于复杂，游戏AI有可能会因为神经网络本身的高复杂性而出现“颤抖”现象。

### 1.3 项目目的

尽管如今支持深度学习的模块有很多，例如tensorflow, keras, pytorch等，但是一方面复杂的数学理论和编程实现会成为大众了解和接触强化学习该前沿领域的门槛。另一方面，它们都缺乏对强化学习在游戏AI方面的特有支持，例如尽管我们可以通过tensorboard可视化网络参数学习过程，但是却没有成型的工具来便捷展示游戏AI的训练分数。另外在实际使用过程中我们也发现，由于训练过程过长，暂停训练并实现重新加载成为开发者迫切需要的功能，但是目前没有统一的平台来支持。第三点，深度强化学习需要学习池储存已经产生的结果用于学习，一旦退出训练过程，保存在内存中的学习池会全部清空，下一次即便加载模型继续学习，也会因为学习池“冷启动”造成前期学习效果不佳，造成时间浪费。

因此我们小组设计出了深度强化学习工具箱，主要用于记录游戏AI在训练过程中每一局的表现并通过服务器实时更新，便于其他开发者在自己设备上查看结果；实现训练项目化，即将每一次训练视为一次项目，如需暂停训练，工具箱可以自动保存此次训练的参数设置、测试游戏、训练状态等，便于开发者下一次重新加载项目文件继续训练过程；统一化参数调节等功能，并提供统一框架，开发者只需要按照命名要求将自己的游戏放入并进行相应的路径设置，工具箱可以自动识别并学习新游戏。希望能够为其他学生接触和学习强化学习提供便利。

# 理论基础

算法来自于DeepMind 在NIPS上发表Playing Atari with Deep Reinforcement Learning 一文，提出了DQN（Deep Q Network）算法，实现通过屏幕学习玩Atari游戏，即只有像素输入，看着屏幕玩游戏。该算法是第一个将深度学习模型与增强学习结合在一起从而成功地直接从高维的输入学习控制策略。具体是将卷积神经网络和Q Learning结合在一起。

### 2.1强化学习是什么？

强化学习属于有监督学习，它考虑的是一个具备行为能力的个体与环境之间的交互任务。强化学习最好的应用场景之一就是游戏AI的训练。在游戏中，玩游戏的人看到的屏幕是环境，人通过键盘、鼠标的操作来控制角色的行动，又产生了新的环境。从上例可以看出，任何一个交互任务，都包含了一个序列，即：动作(a)，观察(s)，反馈值(r)。

* *动作的定义是玩家的操作，如：上下左右*
* *观察的定义是人们尝试着从环境中捕捉到的信息，也称为状态*
* *反馈值的定义为执行的动作与环境交互后，环境发生的变化是好还是坏，如：游戏得分的反馈值就是正的。*

因此，可以发现反馈值是一个量化的标准，反馈值越大表示执行得越好，那么任务的目标就是获取尽可能多的反馈值。可以发现，在每一个状态下应该采取什么动作这是一个决策的过程。而我们一开始并不知道最优的策略是什么，只能从随机决策开始，得到一系列的状态、动作、反馈，成为研究的样本。强化学习的算法就是根据这些样本来改进决策的方法，从而使得序列当中的反馈值更好。

### 2.2 MDP马尔科夫决策过程

绝大多数强化学习都可以模型化为MDP马尔科夫决策过程的问题。马尔科夫决策过程简单来说就是下一个状态仅仅取决于当前的状态和当前的动作。假设每个动作都是由完整的环境决定的，那么只要有一个初始状态，后继状态就是全部确定的。

当把强化学习模型化为MDP问题后，每个动作的概率就像链条中的一个环，被上一个状态确定，同时确定下一个状态，这就意味着每个状态都可以用一个值来描述，并借此判断其状态的好坏，如向左边走能吃到食物，就是好的状态。那么这个值又如何衡量呢？这个值等价于对未来所有时期的回报之和的期望，因此定义一个跨期的（λ代表时间折旧因子0<λ<1，这里假设未来的回报不如当下的回报）回报价值函数：

但是，在这个函数下，如果整个过程没有结束，将无法获取所有的反馈值来计算出每个状态的回报，因此引入估值函数v(s)来表示一个状态的未来潜在价值——表示为跨期回报价值函数的条件期望，该函数可以用来计算已知的某一状态将来预期会带来的回报的总价值。

### 2.3 Bellman方程

那么应该如何求解上述方程呢？通过化简，得到了

**即：**

这是Bellman方程的基本形态，可以通过值迭代的方法计算

### 2.4 优化估值函数：最优动作估值函数

以上我们已经讨论过如何建立、求解估值函数了，接下来将进行优化。2中提到的估值函数中的回报只是某个状态下的估值，而每个状态下都有不同的动作，如果知道某个状态下每种动作的估值，那么就可以选择估值最好的动作去执行。因此定义一个增加了变量a的动作估值函数：

最优可以是找最优的函数、参数、模型。最常用的是找最优的Q值。为了找最优的Q值，将最优的动作估值函数写成：

即选取最优的a’使得Q取最大。

### 2.5 优化迭代方法：Q learning

Q learning的核心是Q-table。Q-table的行和列分别表示状态s和动作a的值，Q-table的值衡量当前状态采取行动a到底有多好。一般的求解方法是通过Bellman方程去更新Q-table，每次根据新的反馈值和原来的Q值来更新Q值：

但是，上述方法将每次都对所有的Q值进行更新，也就是所有的状态和动作，事实上我们无法遍历所有状态和动作，我们只有有限的样本，因此只能使用有限的样本操作，并通过迭代法算出目标Q值，用渐进方式朝目标迈进，使得最终收敛到最优的Q值。

### 2.6 如何生成动作？

在训练过程中，动作该如何生成呢？我们需要使用某个策略生成动作。最常用的有两种策略，一种策略随机生成动作，又称为探索性的策略；另一种策略生成效用最大（基于现有能力）的动作，又称为利用性的策略。第一种策略有利于拓宽模型学习的范围，第二种策略有利于得分的提升。在该算法中动作生成的策略结合了以上两种策略，称为ϵ策略。学习过程中有ϵ的概率生成随机动作，（1-ϵ）的概率生成贪婪性动作。

### 2.7 深度学习神经网络

由于游戏屏幕中像素点很多，状态空间非常大，无法遍历所有的情况，因此如果能用一个函数拟合出估值函数，那么相近的状态就能得到相近的输出动作，因此我们可以通过更新下面函数的参数ω来使Q函数逼近最优的Q值，这可以用监督学习的方法实现。

[Step 1]: 用一个深度神经网络作为Q值的网络，参数为ω，由于我们目标是要实现端对端的控制，即以图像作为输入，因此采用CNN神经网络作为我们参数的结构。

[Step 2]: 根据QLearning理论，我们最终要拟合出一个准确的估值函数Q。当Q函数准确时，我们可以知道

其中。因此拟合估值函数Q就可以等价为降低和之间的差。即优化模型为：

因此最终在Q值中使用均方差来定义目标函数也就是损失函数L。

可以看到，这里就是使用了Q-Learning要更新的Q值作为目标值。有了目标值，又有当前值，那么偏差就能通过均方差来进行计算。

Step 3：计算参数w关于损失函数的梯度

Step 4: 使用SGD随机梯度下降算法（mini batch算法）加快下降速度，即理论上应该使用全体样本得出下降梯度，但此处我们以从全体样本中随机抽取的小样本得到的梯度来近似理论梯度。（因此得名“随机”梯度下降算法）

有了上面的梯度，而可以从深度神经网络中进行计算，因此，就可以使用SGD 随机梯度下降来更新参数，从而得到最优的Q值。

### 2.8 神经网络算法讲解：

### 2.8.1 DQL.py：

#### 初始化：

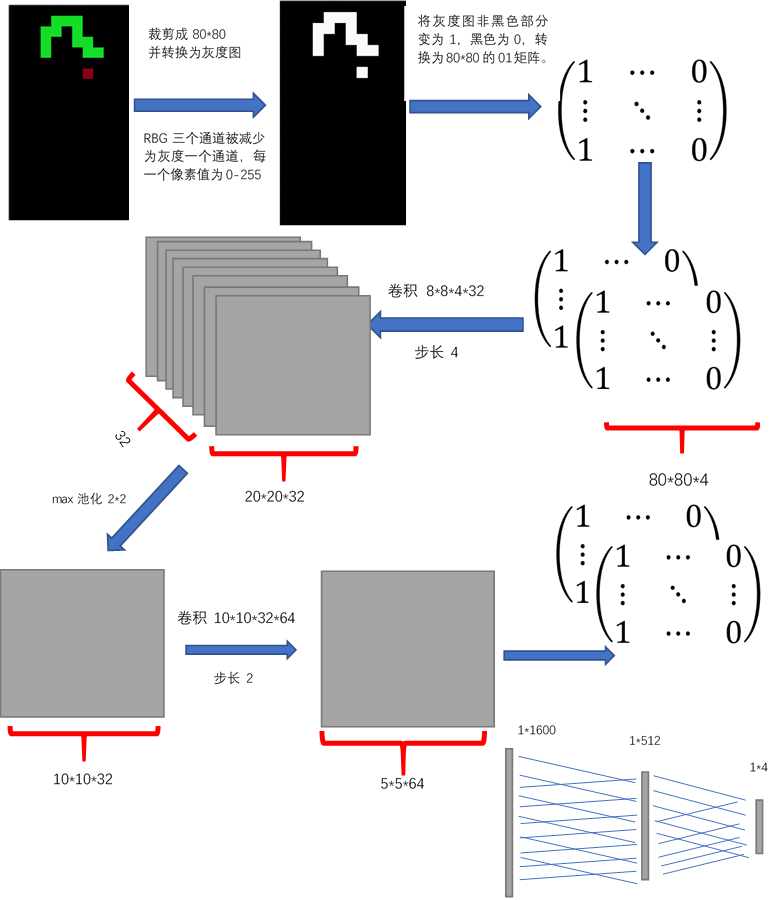
1. 连接数据库
2. 初始化游戏界面
3. 根据选择游戏导入相应的游戏库（避免导入多余游戏造成程序启动速度慢）并传入已初始化的游戏画面来初始化游戏对象
4. 根据设定窗口传入设定数值来初始化强化学习内核。

训练过程中首先给定一个初始化动作传入游戏，生成初始状态（初始观测\*4叠为一个张量）

#### 循环过程：

1. 根据新的状态从内核获取“最优”动作
2. 将新的动作用于输入游戏，得到下一阶段的状态、奖励和游戏是否结束的标志
3. 将新观测进行预处理后放入内核中进行学习
4. 如果该局游戏已经结束，那么将该局分数写入临时数据库中

### 2.8.2 DQLBrain算法详解：

* **神经网络主要由三部分组成：输入层，隐藏层和输出层。**
* **【输入层】**：输入的样本是将游戏相邻4帧界面叠为一个张量，即80\*80\*4的张量。（只采用一帧画面无法表现游戏中的速度等动态变化）。但游戏界面为RGB图片，一共有R、G、B三个通道，每一张图尺寸为512\*288，因此如果直接作为样本输入，那么状态空间会非常大。其实颜色对于某些游戏的进行并不是重点，同时游戏本身如果颗粒度较大的话，也不需要使用高像素画面进行表示。因此输入的样本首先要进行预处理，在DQN\_Module中的preprocess函数。
* **【隐藏层】**：我们基于DeepMind发表在Nature上的神经网络结构，包括了第一层8\*8\*4\*32的卷积层，步长为4；进行一次max pooling降低图片尺寸；第二层为10\*10\*32\*64的卷积层，步长为2；然后进行拉伸变成1\*1600的向量，建立全连接层1600\*512，最后建立全连接层512\*actions(表示该游戏每一帧允许的操作个数)。
* **【输出层】**：输入层和隐藏层完成了局面评估函数，返回值为一个多维向量，表示在当前局面s下各个动作的期望效用，为了得到最终的，由于决策向量为one-hot编码，我们直接将a与G(s)进行点乘，就可以得到在当前局面s下选择动作a的期望效用。完整的结构如下：（以贪吃蛇为例，贪吃蛇每一帧可以选择上下左右四个操作）

### 2.8.3 训练过程：

首先要**【准备训练样本】**，具体实现在setPerception函数中。不同于我们在数据挖掘中比如金融贷款客户违约风险预测等问题，游戏中在相邻时间产生的样本是强相关的，这会严重违反独立同分布的假设，难以达到满意的训练结果。因此这里运用了DeepMind提出的playback策略，即每次从游戏中已有的状态下采取行动，游戏会返回一个新的状态和奖励。因此将作为一个整体存入replayMemory中，训练的时候从其中随机抽取样本来训练。同时配合深度学习中的mini batch算法，即以随机抽取的小样本代替总体来近似总体产生的梯度进行下降。

一旦replayMemory中样本数达到一定限额，那么可以开始**【从中抽取样本进行训练】**。训练过程中先从replayMemory中将四个元素分别抽出形成新的四个列表state\_batch, action\_batch, reward\_batch, nextState\_batch。

根据QLearning**理论**，我们最终要拟合出一个准确的估值函数Q。当Q函数准确时，我们可以知道

其中。因此拟合估值函数Q就可以等价为降低和之间的差。即优化模型为：

在代码中，为了同时存在，我们需要分别保存两个神经网络。因此在初始化的时候我们复制了神经网络，并且当训练次数增加一定额时，将当前的神经网络复制到另一神经网络中，以更新Q网络。

同时为了实现epsilon策略，即每次getAction时以epsilon的概率随机做出动作，以1-epsilon的概率根据目前的Q函数给出当前状态的最优动作，我们采用了[0,1]的随机数与epsilon进行比较，大于则选择最优动作，小于或等于则随机给出动作，以保持状态空间的搜索。

# 测试小游戏



### 跳跳人

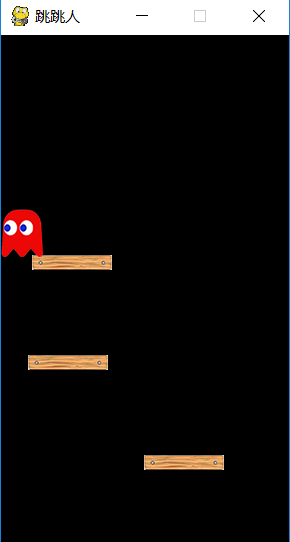
## 3.1.1 涉及模块

Pygame, numpy, itertools, random, sys, time, math

## 3.1.2 游戏规则

玩家通过左右键操作，使小人踩在平板上，得以存活。当玩家碰到屏幕上沿或是掉到屏幕下方，小人死亡，游戏结束。

## 3.1.3 游戏画面



## 3.1.4 模块介绍

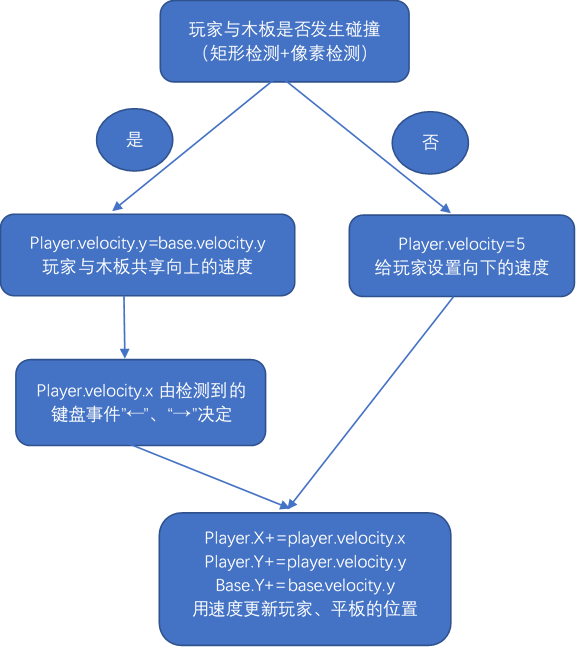
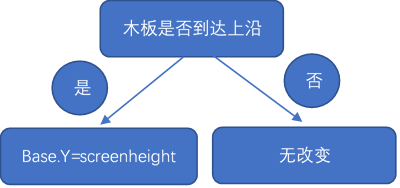
1. 模块my\_library.py：mySprite通过继承pygame.sprite.Sprite获得部分属性，并增加属性velocity、frame等，以及property类型的X、Y，并定义加载图像、更新图像、打印字符等函数。
2. 模块jumpMan.py：主程序，先定义一些全局变量以及玩家和平板的精灵组，再依次定义随机生成板的函数、循环利用板的函数、计算游戏难度同时计算板的速度的函数、计算玩家速度的函数、玩家按左右键时更改图像的函数、以及碰撞检测函数。之后定义了一个JumpMan的类，以类的形式分享私有变量和私有函数等。

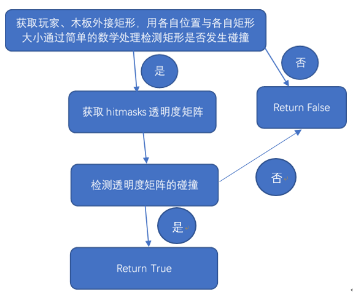
## 3.1.5 设计思路

1. 游戏窗口大小为512\*288，窗口名称为跳跳人。游戏中有两个精灵组，分别管理玩家player和板base，在初始化状态下，在屏幕中央生成玩家精灵，同时在屏幕看不见的下方每间隔100的位置生成5个板，并共享相同的向上的速度。
2. 在固定的关卡内，每一帧每个板都会以稳定的速度向上运动。当玩家没有碰到木板时，将以一个稳定的速度下落。当碰到木板后，将与木板共享一个速度同时向上运动。
3. 每当木板上升到屏幕上方看不见处，重新设置该木板的Y坐标为屏幕下方恰好看不见处。当玩家成功跳下20个木板，游戏难度将加大，木板上升的速度加快。

## 3.1.6 游戏流程图

1. 每帧处理内容



1. 碰撞检测

### 贪吃蛇

## 3.2.1 涉及模块

Pygame, random

## 3.2.2 游戏规则

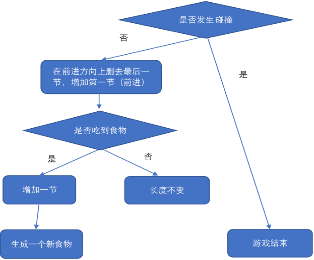
玩家通过上下左右控制小蛇，蛇可以吃到屏幕中随机生成的食物以变长，每当蛇头碰到自身或超出游戏窗口，游戏结束。

## 3.2.3 游戏画面

## 3.2.4 设计思路

1. 游戏窗口大小为512\*288，窗口名称为贪吃蛇。游戏中有两个精灵组，分别管理蛇snake和食物food。
2. 蛇用一个列表表示，当蛇未吃到食物时运动一格，将运动方向的一格添进蛇的身体里，身后的一格从蛇的身体里删除；当蛇与食物发生碰撞时，蛇的长度增长一节，并在除去蛇的身体以外的区域随机生成一个食物。
3. 游戏开始后，玩家可以通过“↑”、“↓”、“←”、“→”控制蛇的运动方向，当蛇与自身或边框发生碰撞时游戏结束。

## 3.2.5 游戏流程图

 （1）每帧处理内容

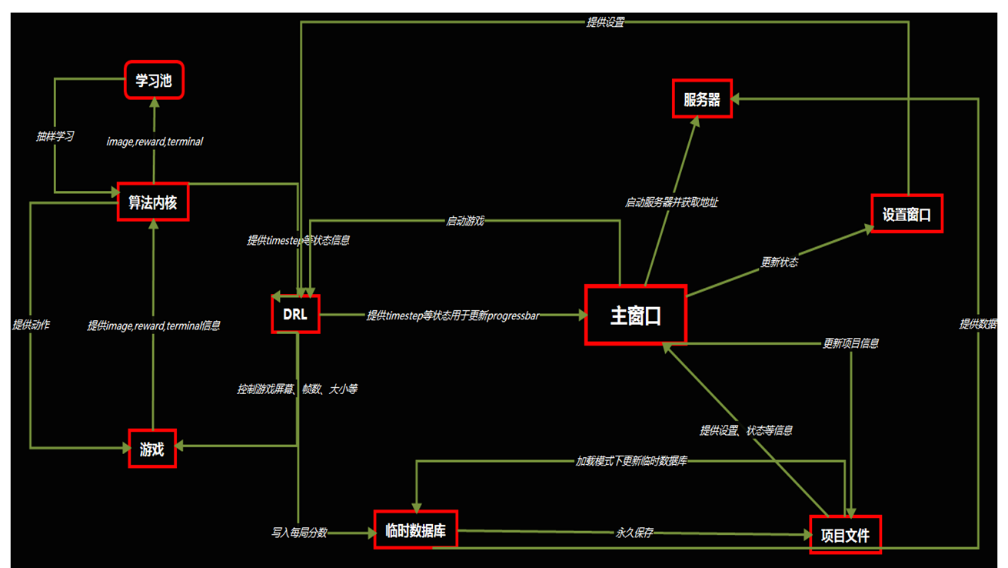
# 工具箱窗口设计

## 4.1涉及模块：

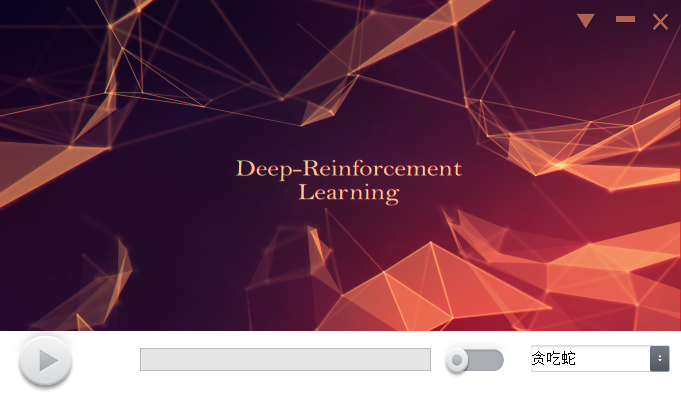
PyQt5、flask\_tk、threading、multiprocessing、sqlite3、pyperclip、shelve、collections、socket、os、flask

## 4.2 设计思路

## 4.2.1 示意图



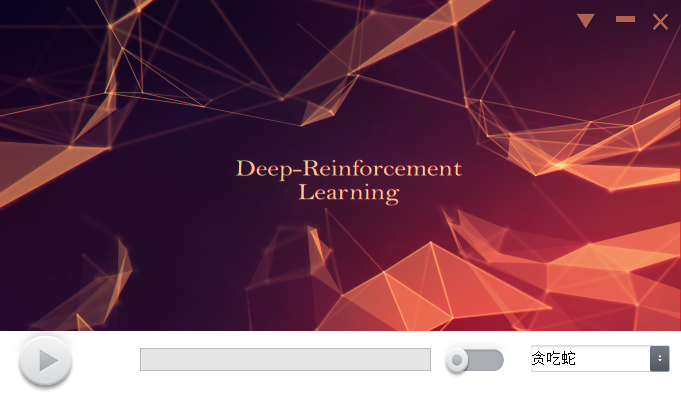
## 4.2.2 主窗口功能详解

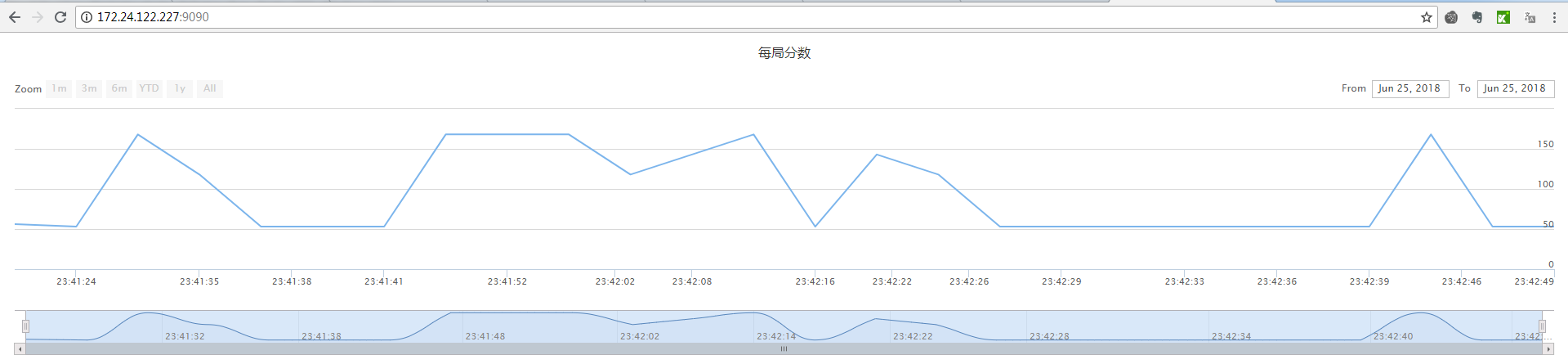
**【设置窗口】**

点击倒三角形状的键，屏幕上会弹出一个黑色的设置窗。在该窗口界面上，用户可以通过拖动进度条、在框内输入具体数值两种方法设置模型参数。

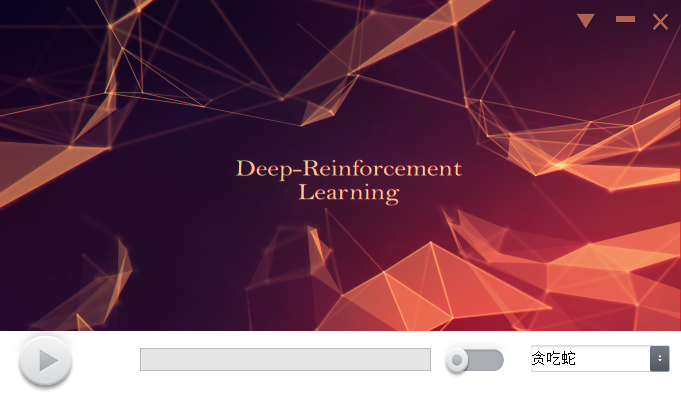


**【在服务器上查看训练结果】**

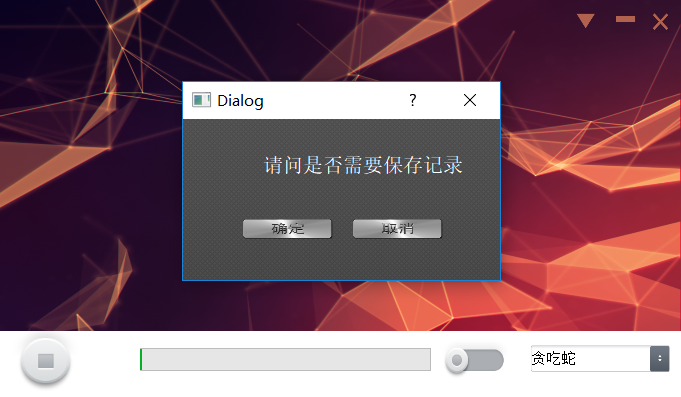


点击最小化按钮，将会复制浏览器地址到剪切板上，可以将其粘贴到浏览器中实时监测训练情况。

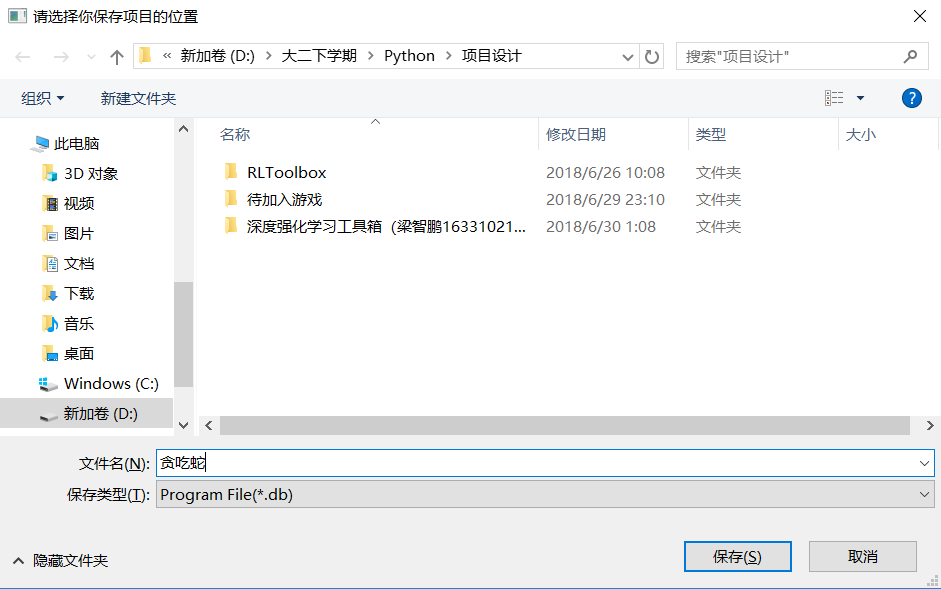
**【关闭按钮】**



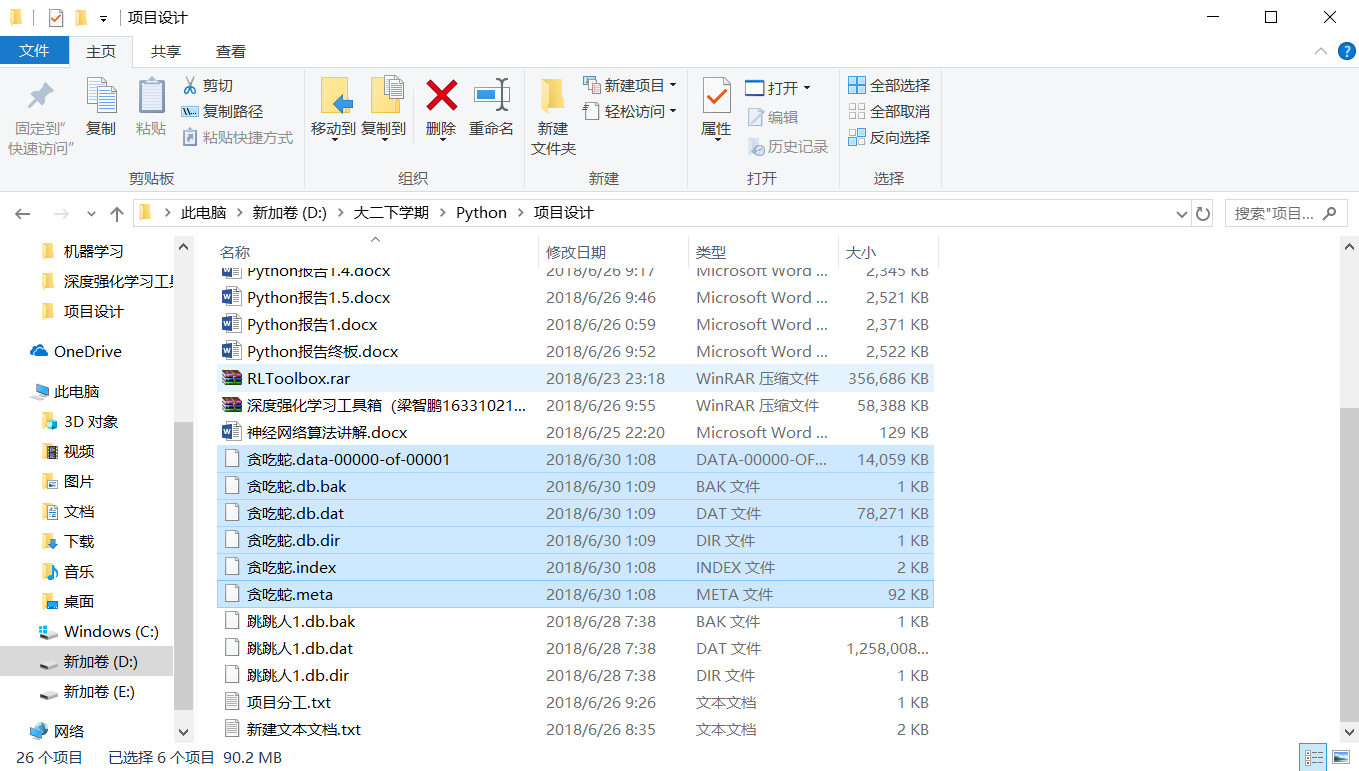
**→当点击关闭按钮时，若训练次数超过1000帧，将会弹出窗口询问是否保存记录。**



**→点击确认**



**→成功保存**

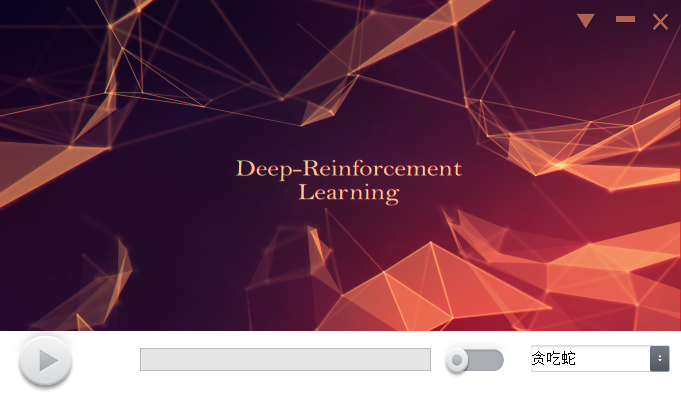


**【新建模式训练】**

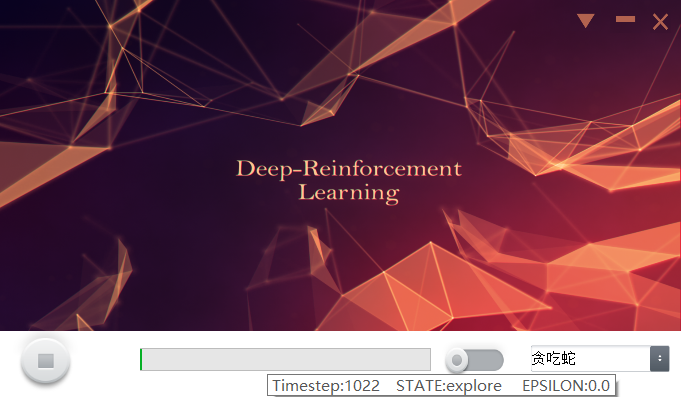
**→选择训练游戏**



**→开始训练**

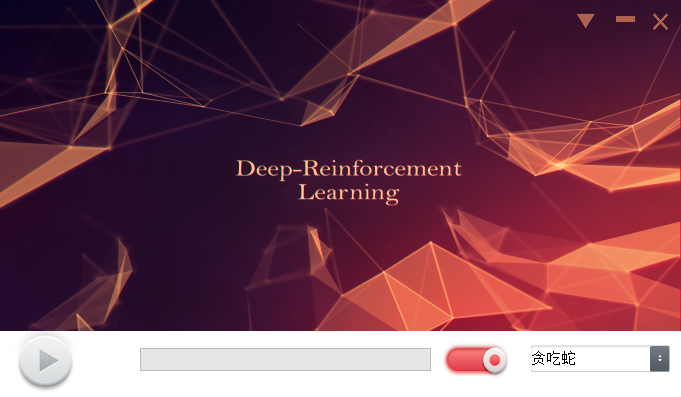


**→鼠标放在进度条上能看到具体数值**

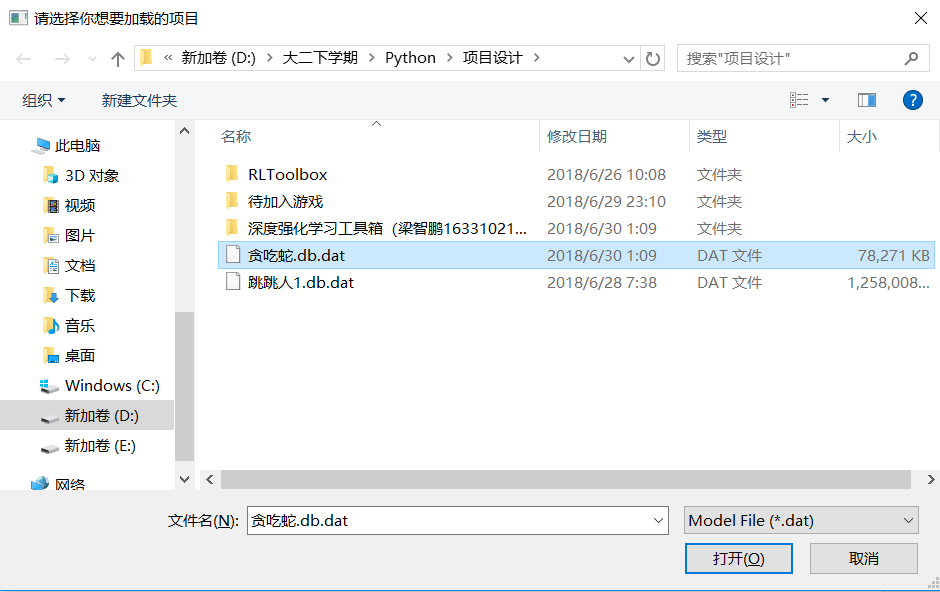


**【加载模式训练】**

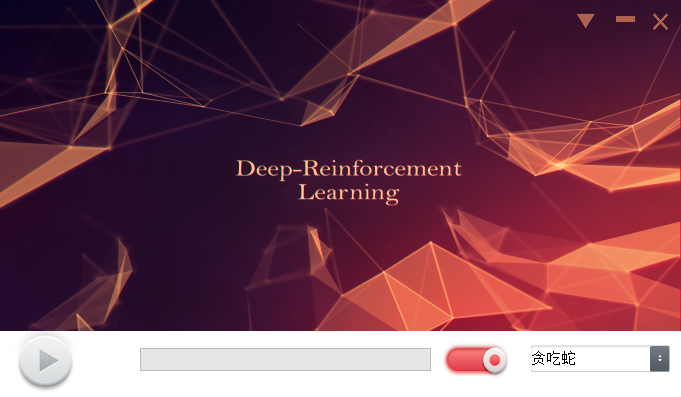
**→点击切换按钮**



**→选择加载模型**



**→点击开始按钮开始训练**



## 4.2.3 主窗口设计思路

1. **【开启】**：主界面分为两种模式：
2. **新建模式：**模式按钮为关闭状态，用户可以点击“设置”按钮，进行参数的调整。也可以通过调整下拉框选择想要训练的游戏。当所以确定完之后，点击启动按钮，为防止干扰训练过程，设置和模式选择按钮被禁用，开启游戏界面并开始训练。
3. **加载模式：**模式按钮为开启状态，无视用户在“设置”中选择的的参数和在下拉框选择的游戏，打开文件选择框。如果用户中途退出，则无后续进程。如果用户选择了已有的项目文件，那么读取项目文件中保存的游戏名称、设置、上次训练状态等并对各状态变量进行更新，直接开启游戏界面训练。
4. **【训练】**：训练过程中用户可以通过进度条的颜色，观察目前训练次数占探索状态空间规定训练次数的比例，还可以通过将鼠标移动到进度条上查看更加详细的信息。AI对象会连接临时数据库temp.db，并且将每次结束时候的分数写入数据库中。
5. **【结束】**：同样基于新建模式和加载模式有两种关闭操作：
6. **新建模式**：将当前的设置、游戏名称、从temp.db临时数据库中抽取训练结果、从内存中读取学习池数据封装成为项目文件，并将训练好的模型从saved\_network文件夹中提取出来，询问用户保存地址和项目名称，进行保存。
7. **加载模式**：按照上述原则更新原有项目文件中的状态和模型。关闭时同时清空临时数据库并断开与数据库的连接、关闭主窗口、关闭pygame窗口和停止服务器等进程和线程。
8. **【设计亮点】**：工具箱将训练项目化，目的在于实现“断点重连”，即记录暂停时候的状态、设置和模型。其中“状态”指的是训练所用的游戏种类，已经训练的次数、学习池的数据、已经得到的每局分数序列。
9. **【进程控制】**：主程序一共有两个进程，第一个进程用于服务器，使服务器每隔3秒从temp.db文件中获取一次数据并实时更新到网页上。第二个进程用于主窗口的响应。其中第二个进程又有一条新线程，用于控制游戏训练过程。画面可自由拖动，另外一个线程用于计时器，每隔5秒完成两个任务：1. 更新进度条及其提示信息 2. 将该5秒内产生的训练结果写入临时数据库中（之所以不是一产生就写入，是因为每次写入很耗时，这种做法会严重影响训练速度）。

## 4.2.4 设置参数菜单设计思路：

**（1） 功能：**

设定八个组件，其中两个是确认按钮和取消按钮。另外有六个互相连接的滑动条和编辑框，可以调整六个训练参数，可以采用窗格内填写数字、拖动进度条两种方式选择参数的具体数值。（进度条有默认最大值，当目标参数超出程序默认最大值时进度条固定在最大处）参数的具体内容如下：

* Explore*：训练次数*
* Initial*：初始犯错率*
* Final*：目标犯错率*
* Gamma*：学习率*
* Replay*：样本容量*
* Batch*：为了进行梯度估计随机选择的小样本*

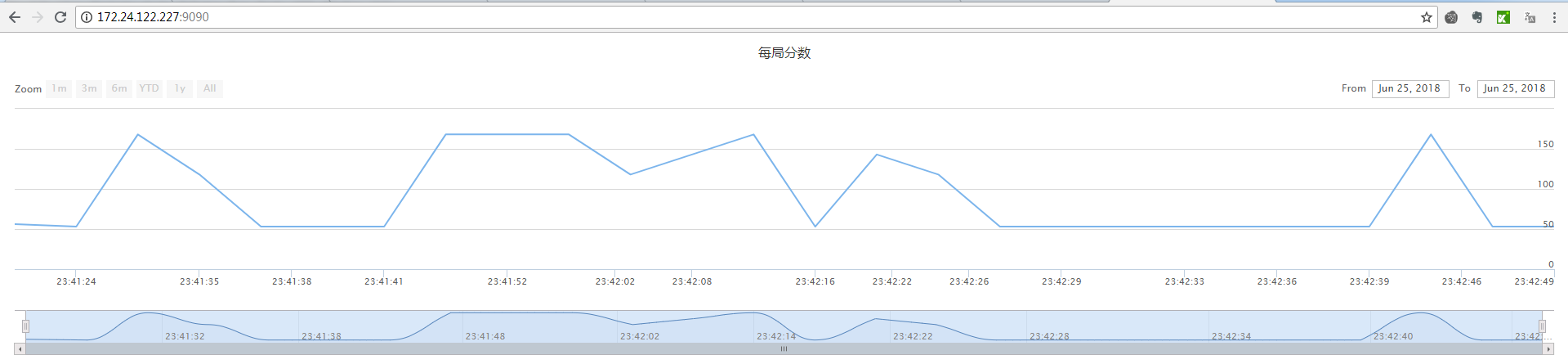
**（2） 设计方法：**

首先进行组件的实例化，再依次定义每个按钮的功能函数。最后通过将组件的点击与各个函数相关联起来并替换组件的样式、背景的样式，最后隐藏默认的窗口。

## 4.2.4 与数据库的交互

（1）每隔五秒，会将训练数据（参数数据与分数数据）保存至一个临时的数据文件当中。

（2）使用者可以通过点击最小化按钮，获得服务器地址，该服务器地址上可以实时观察到训练的成绩。



（3）当关闭时，会弹出窗口询问是否保存训练数据，若选择是，则在指定文件夹中生成可自由命名的数据库文件。

## 4.2.5 服务器设计思路：

使用flask模块，通过设置host ip=’0.0.0.0’监听所有内网ip，使用HighStocks提供的折线图API，写成index.html作为模板。以9090作为通信端口，每隔3秒从temp.db中提取数据并实时绘制在页面中。

# 项目困难及解决方案

**【困难1】**：算法理论复杂，需要长时间理解。

**【解决方案】**：从第五周即开始着手阅读相关论文及材料，从DeepMind发表在NIPS和Nature上的论文入手，将Deep-QLearning细分为QLearning、DeepLearning（CNN神经网络）后分别进行较为系统的学习，并且学习tensorflow语法，咨询老师和有经验的师兄。

**【困难2】**：算法调试困难，需要运行数十万次循环（超过十小时）才能累积出显著的学习提升效果，常用的断点单步调试失效。

**【解决方法】**：可视化损失函数和每局得分以确定程序的正确性。

**【困难3】**：由于GPU擅长并行运算，在神经网络训练过程中速度远超CPU，因此我们选择用tensorflow-gpu，但其配置过程复杂。

**【解决方案】**：查阅Nvida及tensorflow官方文档，并结合网络博客教程进行配置。

**【困难4】**：在建立主窗口以统一监控和管理各模块状态时，变量之间值传递关系复杂。

**【解决方案】**：采用面向对象编程思想，通过类的实例化等方法跨模块引用变量和函数；并且对于各模块需共享的多个变量，我们建立一个字典统一管理。如：在DQLBrain模块中关于timestep、state和epsilon的取值需要同时用于更新主窗口进度条、文件保存等多个功能，通过建立whole\_state字典管理三个变量，在主窗口程序中引用该字典，即可满足变量值的传递。

**【困难5】**：使用PyQt5设计主窗口的过程中，当程序出现错误时不会出现详细的错误信息，只会出现程序崩溃并卡死的状态，导致无法准确定位错误来源。

**【解决方案】**：理清程序逻辑，在尽可能确保没有逻辑上的错误后才启动程序，并通过插入多个输出语句帮助定位错误来源。

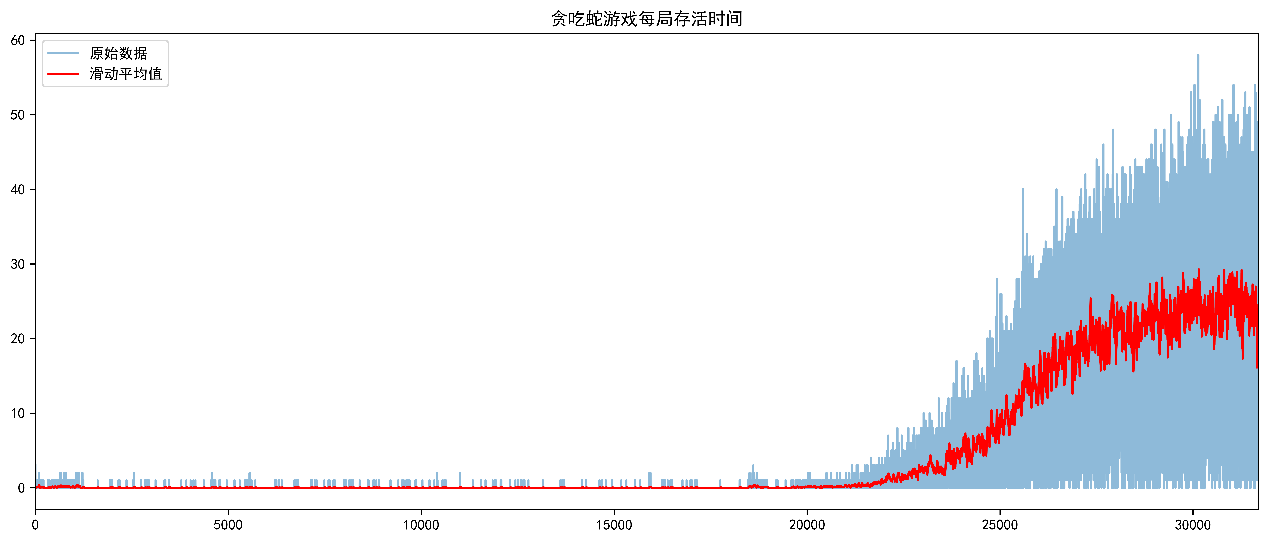
**【困难6】**：服务器、训练器、计时器等多模块并行会导致程序冲突，运行效率低。

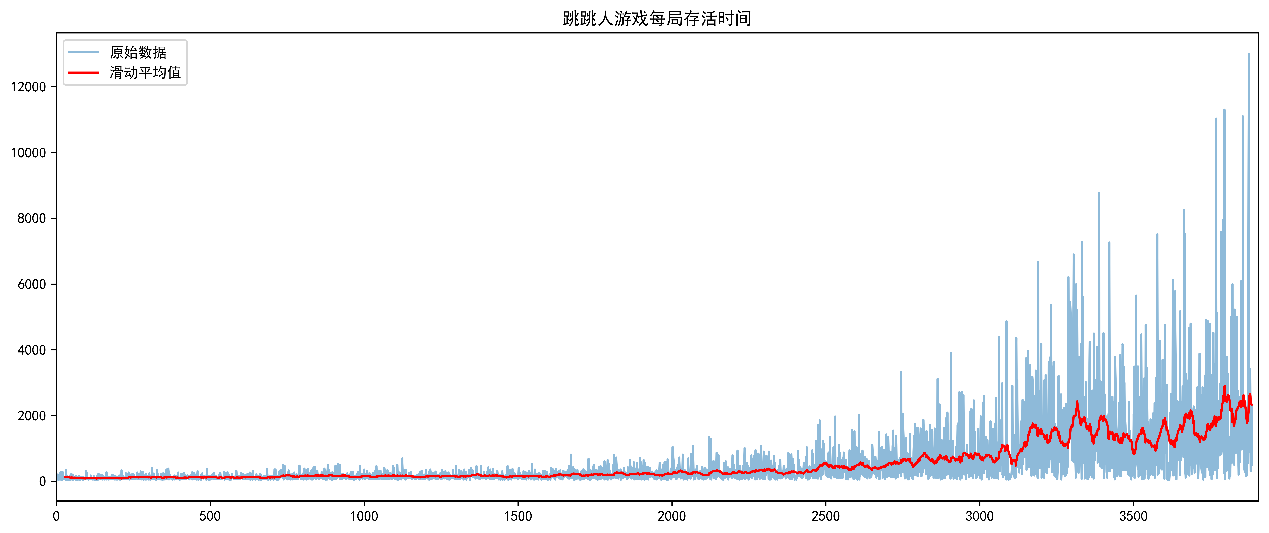
**【解决方案】**：为服务器开设新进程，为定时器和训练器开设新线程，并设置函数统一管理线程的启动和终止。

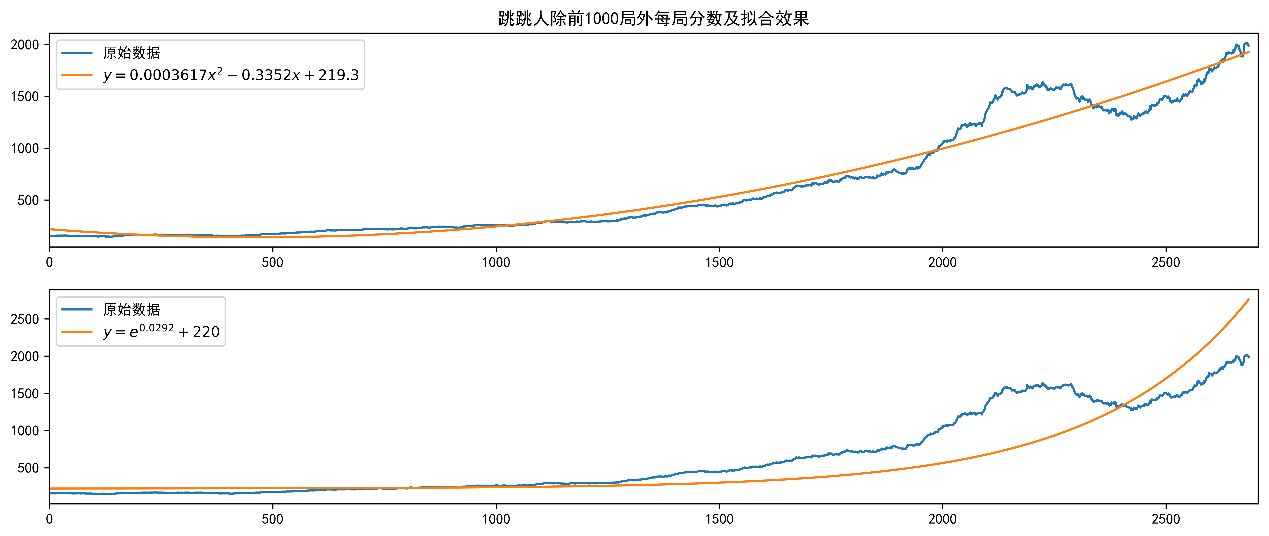
# 项目应用：

团队成员使用编写的项目对贪吃蛇和跳跳人进行了深度强化学习测试。其中用于贪吃蛇的设置详情为：

* Explore*=2,000,000*
* Initial*(起始*epsilon*)=4%*
* Final*(终止*epsilon*)=0%*
* Gamma*(代表折现因子)=0.99*
* Replay*=50000*
* Batch*=32*
* **经过超过500万次训练（超过48小时），一共完成36171局**，每局分数如下图：



* 动态过程可以参考文件中的“贪吃蛇终板.gif”文件。在项目一开始，我们的容错率设置为0，但是随着训练的进行，贪吃蛇会陷入“不断绕圈不吃食物保存自己”的局部最优解策略。
* **跳跳人目前训练了500万次，一共完成3886局，每局存活帧数如下图：**
* **我们对跳跳人的学习速度进行了估计，我们认为从1000局之后跳跳人的学习速度大致符合二次函数以及指数函数**，具体结果如下图：



* 使用同样的参数设置对跳跳人进行训练，同样在经过了超过500万次训练之后，游戏AI有了较为满意的结果。在项目一开始，我们为了加快训练速度，引入了多种激励机制，例如接触到一个新的平台则加分，随着人物所处位置越接近上端，对人物处罚越大。虽然能够迫使人物较快采取积极的措施，但是随着训练的进行出现了“震荡”现象，即人物不断地左右左右移动，影响最优动作判断。根据我们的分析，是由于机制过于复杂，由于神经网络高度复杂，从而导致了对相似情景的不同判断。最后我们仅仅采用了“每一帧+0.5，死亡-5”这样简单的机制，取得了较好的结果。

# 项目分工

### 项目时间进度表：

**第五周**：确定项目方向

**第六周到第十周**：学习强化学习Qlearning基础知识和CNN神经网络基础知识

贪吃蛇游戏编写

**第十一周到第十三周**：深度强化学习代码实现

跳跳人游戏编写

**第十四周到第十六周**：主窗口、设置窗口和消息框窗口编写

报告写作，文件整理

游戏模型训练

### 项目分工：

**梁智鹏**负责深度强化学习、窗体、服务器、数据库的代码编写，报告撰写。

文件包括：

DQL.py，DQLBrain.py，flask\_tk.py，game\_setting.py，mainwindow.py，message\_box.py，run\_window.py，setting.py，index.html

**陈昊**负责窗体、数据库的代码编写，游戏编写，强化学习，报告撰写。

文件包括：

DQL.py，jumpMan.py，mainwindow.py，game\_setting.py，message\_box.py，run\_window.py，setting.py，index.html，MyLibrary.py

**张意伟**负责贪吃蛇游戏编写，报告撰写。

文件包括：

greedySnake.py

### 备注：

Tensorflow 不支持中文注释，因此DQLBrain.py文件使用英文注释。

# 参考文献

[1] Mnih Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rust, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. **Human-level Control through Deep Reinforcement Learning**. Nature, 529-33, 2015.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. **Playing Atari with Deep Reinforcement Learning**. NIPS, Deep Learning workshop

[3] Kevin Chen. **Deep Reinforcement Learning for Flappy Bird** [Report](http://cs229.stanford.edu/proj2015/362_report.pdf) | [Youtube result](https://youtu.be/9WKBzTUsPKc)

[4] Flood Sung. **Deep Reinforcement Learning 基础知识（DQN方面）**<https://blog.csdn.net/songrotek/article/details/50580904>

[5]Giannoccarro Ilaria and Peperpaolo Pontrandolfo. **Inventory management in supply chains: a reinforcement learning approach**, International Journal of Production economics, 2002, 153-161

[6] Zhengyao Jiang, Dixing Xu and Jinjun Liang. **A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem**, <https://arxiv.org/abs/1706.10059>