Flappy Bird 游戏智能体的设计

3220104685 赵一锟

1 实验目的

经过课堂上的学习和课后的阅读，相信大家对强化学习及其相关的 Q-Learning 算法已经有了一定的了解。为了巩固这一阶段学习成果，在本次的 Final Project 中，我们将尝试为 Flappy Bird 这个游戏设计一个智能体，通过强化学习的方式使其能够在这个游戏中获得尽可能高的分数。

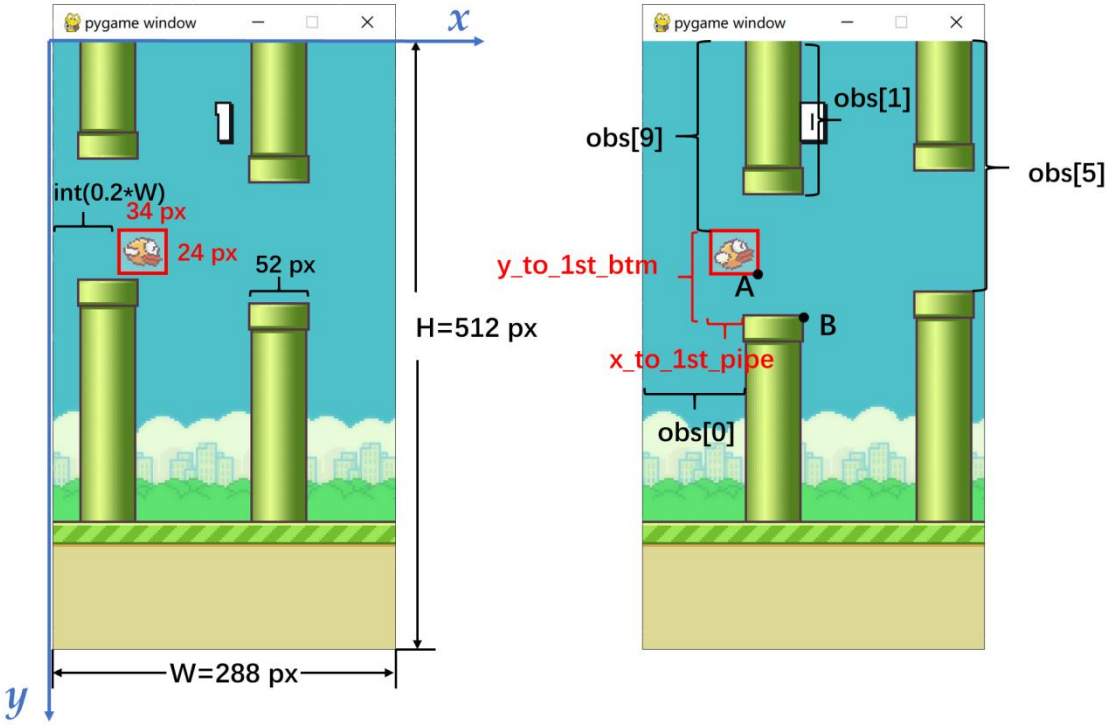

图 1 游戏界面及初始状态设计

2 实验原理

Q-learning 对 flappy bird 游戏的作用

强化学习和 Q-learning 算法通过让智能体在与环境交互中学习最优策略，从而提升 Flappy Bird 的游戏分数。

2.1 强化学习的基本要素

在强化学习中，我们把智能体与环境看作两个交互的系统。

• state

对于 Flappy Bird，state 可以包含小鸟的纵向位置、速度、下一个管道的水平/高度差等信息。在代码中，state 用一个整数列表 List[int] 表示。

• action

智能体在每个时刻可选择的动作：

0：不做任何操作，小鸟自然下落

1：按下空格，小鸟向上扑动翅膀

• reward

环境根据智能体所做的动作给出即时回报。

• policy

智能体根据当前状态选择动作的方式，Q-learning 中的策略由 ε-greedy 决定。

2.2 Q-learning 的核心思想

Q-learning 旨在学习一个动作-价值函数 Q(s, a)，表示在状态 s 采取动作 a 后，遵循最优策略能够获得的最大累积折扣回报。公式为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

其中，α 是学习率，决定新信息与旧估计的权重；γ 是折扣因子，权衡当前回报与未来回报的重要性；r 是执行动作 a 后从环境得到的即时回报；s′是执行 a 后环境转移到的新状态。在代码中，Q-learning 的函数主要与 Game-AI 框架中的 update 函数有关。get_q_value(old_state, action)表示当前 Q(s,a)，best_future_reward(new_state)是 max 项，是通过遍历所有合法动作得到的。

2.3 智能体提升分数的过程：

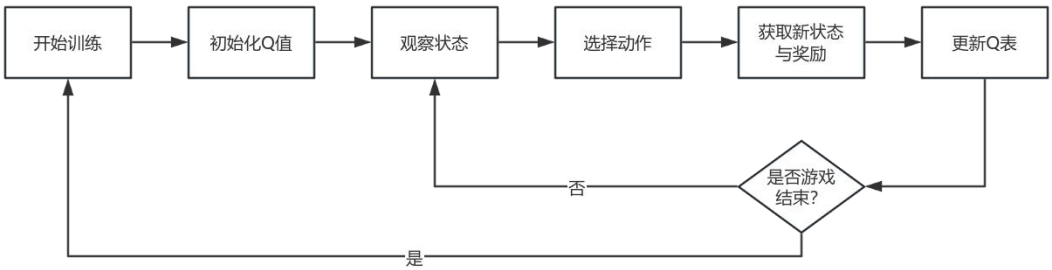

图 2 智能体提升分数的流程图

上图为智能体通过与环境交互获取奖励来进行更新学习的全过程，随着无数次迭代，Q(s, a) 值会逐渐向真实的最优动作价值收敛。通过上述机制，Flappy Bird 的智能体可以不断地从一次次游戏中积累经验，利用 Q-learning 的迭代更新，逐步学会在各种状态下做出最优的决策。最终，Q 函数收敛后，智能体能高效地通过一个又一个管道，最大化游戏得分。

3 实验设计思路与结果比较

3.1 状态的设计

状态设计是对训练模型的上限影响尤为明显的指标。

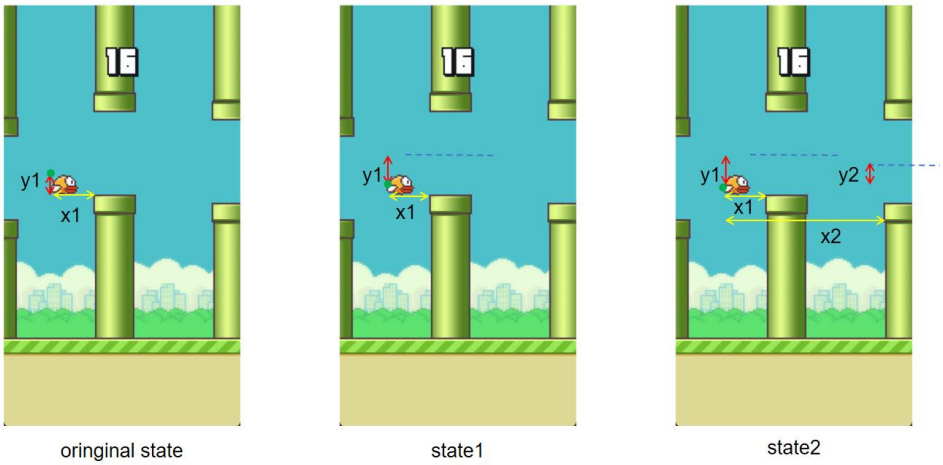

图 3 状态设计变化

在初始的条件（oringinal state）下，我们代表 player 的点处于左上角，有利用到第一个管子的水平距离、到下方管子的垂直距离以及 player 的速度作为状态的三个维度，我们以 player 的右侧为基准判断 player 是否越过第一组管道。在这种条件下，我们可以得到的较优的训练参数及结果是源程序跑出来的结果：32.2

```
(base) PS C:\Users\Yikun_Zhao\Desktop\Junior_2\AI\project\final-project-2025\src> python .\train_ai_or_play.py
This try gets 62 score(s).
This try gets 24 score(s).
This try gets 5 score(s).
This try gets 2 score(s).
This try gets 68 score(s).
The average score(s) of Q-Function: 32.2
```

图 4 初始训练结果

• state1 设计思路

我们首先可以观察到，以 player 的右侧作为基准判断其是否越过第一组管道是不合理的，因为这并没有考虑到 player 的身长，因此我们需要调整这一策略，避免智能体在这一层面的误判，改用 player 右侧作为基准判断其是否通过第一组管道。

与此同时，我们可以观察到 player 的垂直基准是下管道的顶部，虽然其可以作为基准，但由于碰到管道就会游戏结束，因此很明显它只允许向上的误差，不允许向下的误差。对此，我们将基准调整至一组管道的中线，这样在训练的过程中，player 可以在基准线上下波动寻找位置，可以在相同训练次数的情况下节约训练所需的迭代次数。

如图 3 的 state1，在以上两种修改结束后，我们可以在原参数的基础上进行训练，得到较好的训练结果为平均 379.2。可以看到，在修改与优化状态之后，模型的分数提升显著。

```
This try gets 260 score(s).
This try gets 458 score(s).
This try gets 805 score(s).
This try gets 53 score(s).
This try gets 320 score(s).
The average score(s) of Q-Function: 379.2
```

图 5 state1 训练结果

• state2 设计思路

通过观察给定参数，我们不难发现：下下组管道的位置我们并没有充分利用。因此，我引入到第二组管道的中心的水平距离与垂直距离。初步引入后，由于状态数的增加，不确定性显著增加，我发现 player 需要经过很长时间的训练才能让 q 值逐步收敛，实验的分数逐渐提高。初步训练时得到的结果较差，仅为 14.4

```
This try gets 33 score(s).
This try gets 6 score(s).
This try gets 2 score(s).
This try gets 11 score(s).
This try gets 20 score(s).
The average score(s) of Q-Function: 14.4
```

图 6 state2 初步训练结果

在增加两个维度的信息的基础上，理论上 player 与的环境交互结果应得到更加准确的游戏模型。因此结合后面的调参过程与修改 reward 的过程，我们最终得到了 Q 值收敛较快的模型，训练结果达到了 7078.8，这也是本次实验的最优模型

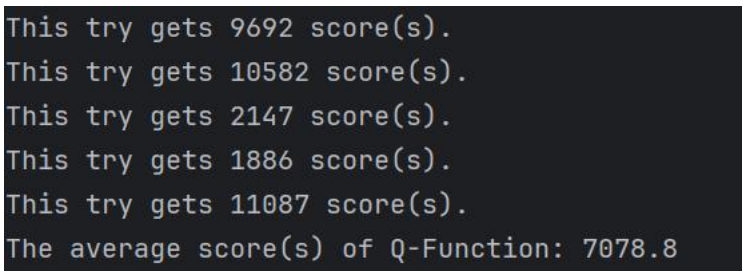

图 7 state2 最佳训练结果

3.2 模型参数调整

3.2.1 不同参数可能对实验结果产生的影响

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

• 学习率 alpha

学习率 alpha 主要控制 “新信息”与“旧估计”。当 alpha 过大时，智能体几乎“只信”最新的回报，快速跟新，却可能把偶然的高分或低分信号过度放大；当 alpha 过小时，智能体更依赖初始估计，学习“动作价值”更新太慢，需要极多游戏局数才能看到明显进步。

• 折扣因子 gamma

折扣因子的主要作用是衡量“未来奖励”的重要性。当 gamma 靠近 0 时，智能体只追求眼前“过管道得 +1”或“撞管道 −1000”，不懂“管道序列的综合价值”，因此容易做出极端动作，只管短期生存；当 gamma 靠近 1 时，智能体过度关注未来长远收益，但若游戏长度无限制，累积回报可能爆炸或训练不稳定。而 flappy bird 没过一关回报较小，gamma 较大值时能够权衡未来与即时的利益。

• 探索率 epsilon

在 ε-greedy 策略中，以 ε 概率随机探索，以 1 − ε 概率利用当前 Q 表中的“最佳”动作。epsilon 太大时，分数大起大落，利用已有好策略的机会太少；epsilon 太小时则基本不探索，一旦陷入局部最优很难跳出。

• 迭代次数 iteration

迭代次数是实际运行多少次游戏回合，用于不断更新 Q 表。当迭代次数太少时，Q 表中大量(state, action)仍未访问或访问太少，无法获得可靠估计；若迭代次数过多，如果参数选择不佳，可能收敛到次优解。

• obs_mul_factor

obs_mul_factor 在状态处理里的核心作用是将游戏环境中连续或大范围的观测值（如管道间距、小鸟速度等）映射到一个可控的、相对离散化的整数网格上，从而决定了“状态空间”的精细度和规模大小。其值越大，连续观测被放大后四舍五入为整数，离散化后状态的区分度就越高，即使小的变化也能映射到不同状态上，使智能体能“感知”更细微的环境差异，但需要更多样本去“填满”庞大的 Q 表，收敛慢、样本效率低；其值越小，状态被压缩得更粗，许多相近的观测会合并到同一个状态里，智能体只能以更“模糊”的视角决策，可能错过细节优化，最终得分上限受限。

3.2.2 参数调试阶段（包含 reward）

注：参数调试部分仅展示了调试不同阶段的部分较为有效的结果，大量繁杂、效果差的结果已省略。

第一阶段：控制变量调整阶段（仅选取部分较好的结果展示）

• alpha　obs_mul_factor = 30

alpha	gamma	iteration	epsilon	result
0.7	0.95	50000	0	32.2
0.6	0.95	50000	0	14.8
0.8	0.95	50000	0	7.8
0.9	0.95	50000	0	24.4

• gamma　obs_mul_factor = 30

alpha	gamma	iteration	epsilon	result
0.7	0.95	50000	0	32.2
0.7	0.9	50000	0	14.8
0.7	0.94	50000	0	21.2
0.7	0.96	50000	0	18.0

• epsilon　obs_mul_factor = 30

alpha	gamma	iteration	epsilon	result
0.7	0.95	50000	0	32.2
0.7	0.95	50000	1e-6	12.2
0.7	0.95	50000	1e-4	3.0
0.7	0.95	50000	0.1	0.0

obs_mul_factor

alpha = 0.7, gamma = 0.95, iteration = 50000, epsilon = 0

obs_mul_factor	result
10	4.8
20	4.4
30	32.2
50	14.4

通过控制变量的方式调整参数，我们发现调整之后的参数得到的分数都不如原有的参数更好，因此仅调整一个参数时，原有的参数组合是最佳的。

不过控制变量调参本身忽略了不同参数之间的相互左右。比如 alpha 与 gamma 的组合就会影响强化学习的策略：

高 γ + 高 α：快但不稳；

高 γ + 低 α：稳但慢；

低 γ + 高 α：极短视且躁进；

低 γ + 低 α：既短视又迟缓。

第二阶段：多参数联调阶段：

因此，针对这两个参数，我们对其进行不同程度的变化，选取在初始状态下较好的结果记录下来

alpha and gamma

alpha	gamma	iteration	epsilon	result
0.7	0.95	50000	0	32.2
0.72	0.9	50000	0	12.8
0.5	0.98	50000	0	36.4
0.3	0.99	50000	0	26.8

在这次实验中，我们可以知道联调的确可以让 result 的结果有变化，并且 0.5 0.98 的组合比初始的 0.7 0.95 的组合要更好。

第三阶段：探索最佳组合

• iteration 的选择

由于不同的参数变化与最终的结果变化不是线性、同方向的，不同状态的最佳参数也各不相同，我们很难将所有参数经过穷举得到一个最终收敛的最佳结果。因此我们将 iteration 作为参数进行调整。在此过程中，我们首先在训练环节加入定期的 test，使其能够一边训练，一边可视化出 result 随 iteration 的变化。这样我们就可以在特定 Iteration 下，得到所调整到的参数的最佳组合。本项目使用 100 作为步进调整 iteration，具体可视化代码如下：

```
        # 定期测试
        if (i + 1) % 100 == 0:
            avg_score = test_model(player)
            print(f"Iteration {i + 1}, Average Test Score: {avg_score}")
```

```
def test_model(player, num_games=5):

    env = gymnasium.make( id: "FlappyBird-v0", render_mode=None, use_lidar=False)
    scores = []
    obs, _ = env.reset(seed=42)

    for _ in range(num_games):
        obs, _ = env.reset()
        while True:
            action = player.choose_action(process_obs(obs), use_epsilon=False)
            obs, _, done, _, info = env.step(action)
            if done:
                scores.append(info['score'])
                break

    env.close()
    return sum(scores) / len(scores)
```

图 8 定期测试代码

引入以上代码，我们就可以实时精准地测试 iteration 的最佳位置。如下图所示，我们可以看到在 22000 次 iteration 左右，训练的分数要远高于迭代次数减小或增大后的结果。通过这种方式，我们可以实时调整 iteration 来调整获得训练的最佳结果。

```
Iteration 21500, Average Test Score: 87.4
Iteration 21600, Average Test Score: 87.4
Iteration 21700, Average Test Score: 87.4
Iteration 21800, Average Test Score: 87.4
Iteration 21900, Average Test Score: 118.2
Playing training game 22000
Iteration 22000, Average Test Score: 118.2
Iteration 22100, Average Test Score: 87.4
Iteration 22200, Average Test Score: 87.4
Iteration 22300, Average Test Score: 87.4
```

• reward 的调整

reward 的调整策略：1 强化撞死后的惩罚力度。如果撞死结束的惩罚太轻，智能体可能在多次小失败中依然沉浸在随即探索，难以学会避障。2 适当提高过管道的正向激励，这样智能体更倾向于那些能稳定过管道的动作组合，平均得分在较少迭代内迅速攀升。

由于 reward = -1 与 reward = -0.5 的情况中，撞天花板与撞墙/管子均会导致游戏结束。因此我们将其均改为 reward = -1000，在此基础上，将原来的通过管道加分与存活加分均提升指原来的 10 倍。

具体调整的过程与调整的状态设置有关，不同状态空间的背景下，不同 reward 可能产生不同的结果。我们仅以最佳结果为例：

reward 为（-1000，-0.5，1，0.1）时：

```
This try gets 2 score(s).
This try gets 1 score(s).
This try gets 5 score(s).
This try gets 3 score(s).
This try gets 3 score(s).
The average score(s) of Q-Function: 2.8
```

reward 为（-1000，-1000，10，1）时：

```
This try gets 9692 score(s).
This try gets 10582 score(s).
This try gets 2147 score(s).
This try gets 1886 score(s).
This try gets 11087 score(s).
The average score(s) of Q-Function: 7078.8
```

我们可以明显地观察到，由于 reward 设置的原因，初始状态的 player 几乎都是向上飞的过高而撞死，这是由于撞到天花板惩罚太小导致的，因此 reward 对结果的影响也十分显著。

3.3 best_test 展示

reward (-1000, -1000, 10, 1)

alpha 0.5

gamma 0.98

epsilon 0

iteration 22000

obs_mul_factor = 30

```
This try gets 9692 score(s).
This try gets 10582 score(s).
This try gets 2147 score(s).
This try gets 1886 score(s).
This try gets 11087 score(s).
The average score(s) of Q-Function: 7078.8
```

4 总结与收获

通过本次基于 Q-learning 的 Flappy Bird 实验，我们实现了使用强化学习来完成 flappy bird 小游戏的智能体。通过参数的调整与训练，我们的最佳模型平均分数达到了 7078.8，效果良好。在实验的过程中，除了进一步熟练了 python 的使用之外，我还了解了调参对于强化学习的重要性，参数的调整既要结合理论又要通过实践去不断试验，有时候很差的结果仅仅需要改动一个参数就能得到大幅度的提升。因此在此过程中我们不能主观臆断，要更多地用实践去验证自己的想法。本次大作业让我受益匪浅，希望在未来人工智能相关的学习与实践中能够继续努力、不断收获与成长。