國立臺南大學資訊工程學系

109 級畢業專題報告

基於強化學習之格鬥遊戲即時反應操作 AI 設計

AI of Fighting Game

專案編號：NUTN-CSIE-PRJ-109-020

組員：S10559028 黃仁鴻

指導老師： 陳榮銘 教授

中華民國 109 年 1 月

國立臺南大學資訊工程學系
畢業專題報告審定書

基於強化學習之格鬥遊戲
即時反應操作 AI 設計
AI of Fighting Game

學生　　學號：S10559028　姓名：黃仁鴻

所提之報告內容符合本學系大學部畢業專題實作標準。

指導教授：_____

系主任：_____

中華民國 109 年 1 月

基於強化學習之格鬥遊戲即時反應操作 AI 設計

摘要

一般格鬥遊戲上，常有入門門檻過高與玩家間常出現嚴重的技術斷層等等問題，新手玩家連初級 AI 都打不贏，而老玩家面對高階 AI 卻依舊取得壓倒性勝利，這樣巨大的斷層會帶給格鬥遊戲的初學者巨大的挫敗感。

因此，本專題藉由強化學習方法訓練出能用於格鬥遊戲的 AI，並利用調整獎勵來製作出不同風格、強度的模型，使得各種玩家都能充分享受格鬥遊戲帶來的樂趣。

關鍵詞: 格鬥遊戲、深度學習、強化學習。

AI of Fighting Game

Abstract

In general fighting games with artificial intelligence (AI), there are often problems such as high learning difficulty and serious technical faults between the players. Novices can't even win primary AI, but an old hand still has overwhelming victory against high-end AI, which will bring a huge frustration for the beginners.

Therefore, in this independent study, we train AIs for fighting games through reinforcement-learning techniques. The adjustment rewards are used to generate models of different styles and levels, so that the games can interact better with various players and the players can fully enjoy the fun brought by the fighting games.

Keywords: fighting game, deep learning, reinforcement learning

誌謝

首先感謝指導教授陳榮銘老師及實驗室郭偉祥學長的指點與鼓勵，在這一年的研究過程中遇到了許多的挫折及壓力，感謝兩位的經驗幫助我通過難關，並提供優良的研究環境使本專題能夠順利進行。還有友人們的討論與建議皆成為本實驗的重要參考。

於此鄭重感謝所有幫助過我實驗的師長與友人們，各位是本專題成功的重要基石，謝謝你們。

黃仁鴻 謹致

國立臺南大學資訊工程學系

中華民國 109 年 1 月

目錄

表目錄

圖目錄

一、簡介

一般格鬥遊戲上，常有入門門檻過高與玩家間常出現嚴重的技術斷層等等問題，新手玩家連初級 AI 都打不贏，而老玩家面對高階 AI 卻依舊取得壓倒性勝利，這樣巨大的斷層會帶給格鬥遊戲的初學者巨大的挫敗感。

因此，本專題藉由強化學習[1]方法訓練出能用於格鬥遊戲的 AI，並利用調整獎勵來製作出不同風格、強度的模型，使得各種玩家都能充分享受格鬥遊戲帶來的樂趣。

為了方便進行研究以及環境佈署，將會使用 tensorflow.js[2]開發模型，使模型可以在瀏覽器上訓練，並自行製作測試用遊戲，以便設計輸入與控制接口。

二、問題描述

本次研究要利用強化學習設計適合用在格鬥遊戲之代理人(Agent)，讓 AI 可以透過不斷練習使技巧精進，而在完成此目標的研究過程中會遇到以下兩個主要問題：

1. 如何讓代理人判別出最適合當下狀態使動作，使得整場對戰帶來最大效益，為達成此目標，需要定義好輸入、輸出、模型架構與獎勵函數。

2. 模型將運用在與環境快速交互的格鬥遊戲上，因此，需要讓類神經網路能夠快速地計算出結果。這樣的要求就必須要限制模型的計算量與深度。

三、研究方法

圖 1 為本專題的開發流程，將使用強化學習算法中的 Dueling Double DQN[3,4,5]作為主要研究方法，並測試各類獎勵函數，使模型能更適合使用在格鬥遊戲上。

開發流程的分述如下：

1. 藉由 babylon.js 與 blender 於瀏覽器上建置格鬥遊戲環境。
2. 使用 tensorflow.js[2]實作 Dueling Double DQN 後與測試環境進行連結。
3. 讓模型自我對戰並開始訓練。
4. 完成訓練後測試模型強度並調整超參數(包含獎勵函數與類神經網路架構)。
5. 重複步驟 3、步驟 4，直到模型表現良好穩定。

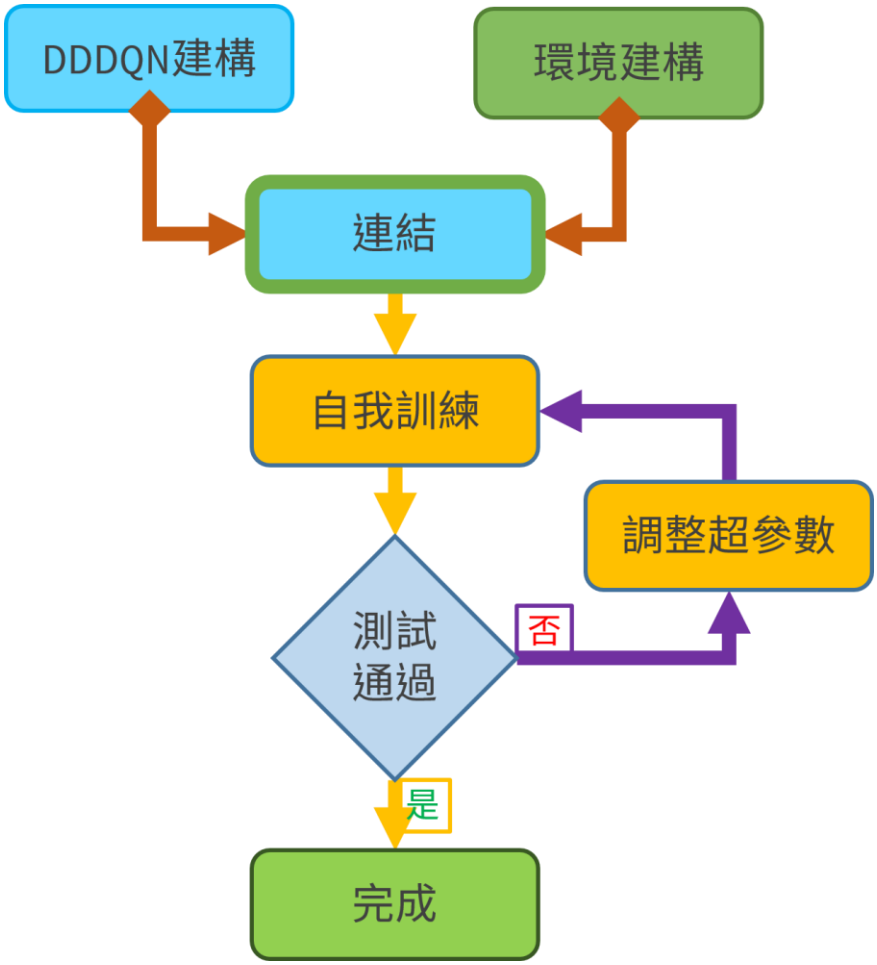

圖 1 開發流程

3.1 格鬥遊戲環境建置

為方便研究之進行，本專題會先製作用於測試環境之格鬥遊戲，格鬥遊戲使用了微軟開發的遊戲引擎 babylon.js 進行製作，表 1 記載著角色各個動作與對應的觸發按鍵，角色動作由<上/下/左/右>與<輕/中/重>攻擊共 7 個按鍵搭配而成。

表 1 角色動作組

動作	按鍵	說明
站立	無	
前進	<右>/<左>(依據對手位置決定)	靠近對手
後退	<左>/<右>(依據對手位置決定)	遠離對手
蹲下	<下>	
跳躍	<上>	可二段跳
防禦	同「後退」	可減輕受到的傷害
完美防禦	再受攻擊前的 0.1 秒內「防禦」	完全擋下傷害
受擊狀態	無	受到攻擊後產生
「站立」<輕>攻擊	「站立」時按下<輕>攻擊	
「站立」<中>攻擊	「站立」時按下<中>攻擊	
「站立」<重>攻擊	「站立」時按下<重>攻擊	執行動作時不會被打斷
「蹲下」<輕>攻擊	「蹲下」時按下<輕>攻擊	
「蹲下」<中>攻擊	「蹲下」時按下<中>攻擊	
「蹲下」<重>攻擊	「蹲下」時按下<重>攻擊	執行動作時不會被打斷
「跳躍」<輕>攻擊	「跳躍」時按下<輕>攻擊	
「跳躍」<中>攻擊	「跳躍」時按下<中>攻擊	
「跳躍」<重>攻擊	「跳躍」時按下<重>攻擊	執行動作時不會被打斷
墜地攻擊	「跳躍」時按下<下>與<重>攻擊	快速向地面下墜

3.2 Dueling Double DQN

DQN[3]是 Deep Q-learning Network 的簡稱，結合了 Q-learning 與類神經網路的一種強化學習算法，並於後來出現多種改進算法，其中就有 Double DQN[4]與 Dueling DQN[5]這兩種改良方式。而本專題採用的 Dueling Double DQN[3,4,5]就是採用 Dueling DQN 的架構，搭配 Double DQN 對目標價值的計算方式所構成的。

圖 2 是 DQN 系列的運作流程圖，類神經網路會將環境目前狀態(S_t)作為輸入以此估算出每個動作的價值(Qs)，藉由 Q 值挑選出要執行的操作(A_t)，執行完成後環境就會產生下一個狀態(S_{t+1})與該操作所得到的獎勵(R_t)，並將$\{S_t, A_t, R_t, S_{t+1}\}$作為一筆記憶存入經驗池中。

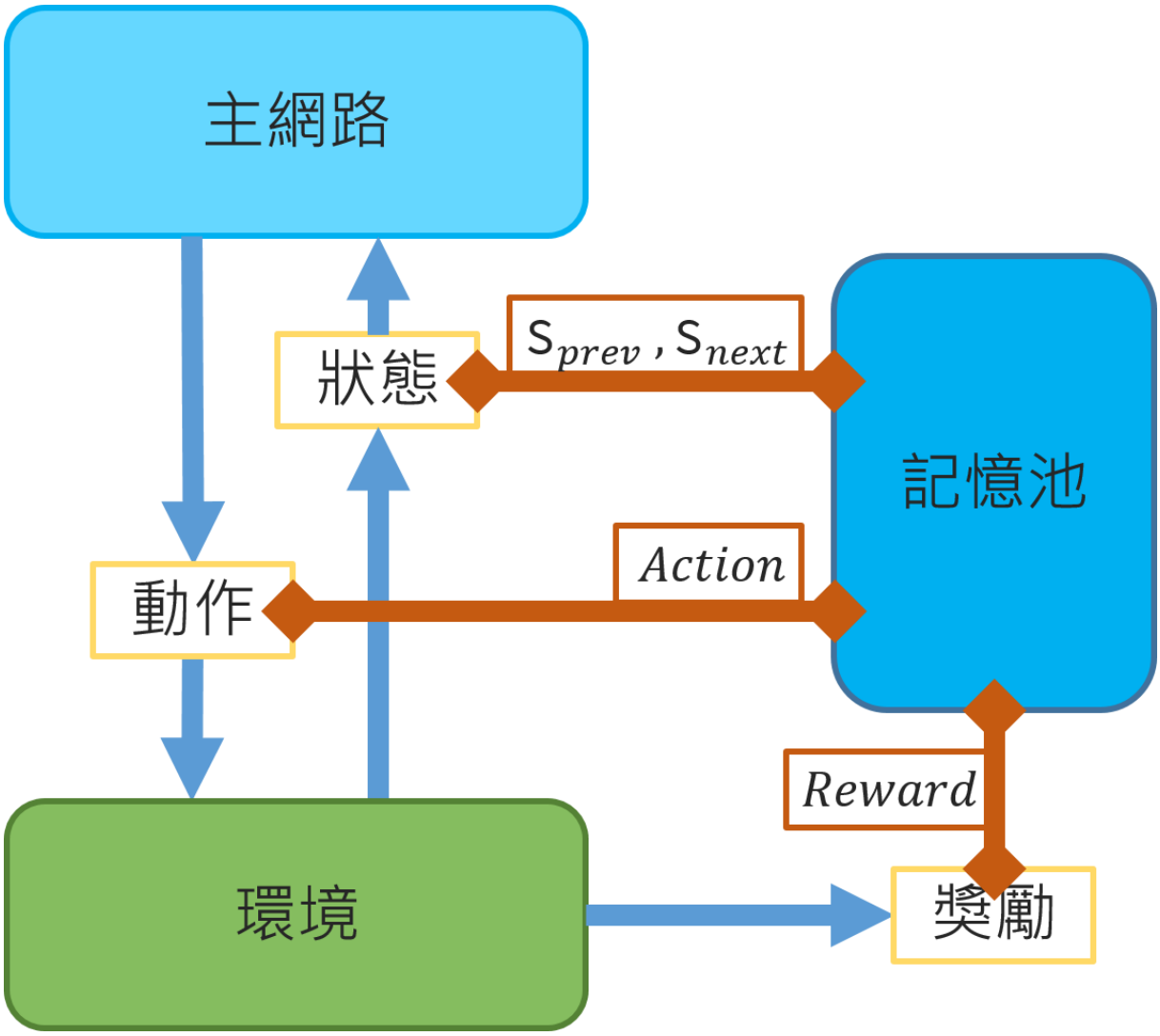

圖 2 DQN 運作流程

1. DQN

DQN[3]具有Net_{main}與Net_{target}兩個類神經網路，在初始化時具有相同的權重，然後從記憶池抽取記憶，一筆記憶的構成為$\{S_t, A_t, R_t, S_{t+1}\}$(S 代表狀態、A 代表執行的動作、R 代表做完動作後的到的獎勵值)，在利用這筆記憶進行目標 Q 值的計算：

$$\text{Qs} = Net_{main}(S_t) \tag{1}$$

$$Q_{target} = R_t + \tau \times \max(Net_{target}(S_{t+1})) \tag{2}$$

$$\text{loss} = \text{mean square error}(Qs[A_t], Q_{target}) \tag{3}$$

從式(1)與式(2)得出Qs與Q_{target}後，如式(3)使用 mean square error 計算出 loss 並更新Net_{main}的權重，當對Net_{main}進行數次更新後，就可將Net_{main}的權重複製給Net_{target}。

若模型在進行預測時計算出的價值接近實際的獎勵(Q ≈ R)，則式(2)中的$\max(Net_{target}(S_{t+1}))$便可以代表在狀態$S_{t+1}$時能獲得的最大價值$R_{t+1}$。且在更新權重後會使$Net_{main}(S_t)[A_t]$接近$Q_{target}$，當選擇價值最大的動作時以上可以理解成式(4)：

$$\max(Net_{main}(S_t)) \approx R_t + \tau \times R_{t+1} \tag{5}$$

當Net_{main}將權重複製給Net_{target}後，$\max(Net_{target}(S_{t+1}))$就等同$\max(Net_{main}(S_{t+1}))$近似$R_{t+1} + \tau \times R_{t+2}$，因此式(2)可以更新成：

$$Q_{target} = R_t + \tau \times (R_{t+1} + \tau \times R_{t+2}) \tag{6}$$

從上述可知，當經過無數次更新後再將Q_{target}展開便得出：

$$\sum_i R_{t+i} \times \tau^i \tag{7}$$

由式(7)可知估測價值會與未來可取得的最大價值相關，而不會只一昧地著重在眼前的獎勵，其中τ代表折扣係數，$0 \le \tau < 1$，藉由調整其大小來改變未來獎勵對目前決策的影響。

2. Double DQN

Double DQN[4]與 DQN[3]的差別在於目標 Q 值的計算方法，Net_{target}本身預測會有誤差，然而式(2)的算法會選擇$Net_{target}(S_{t+1})$預測出的最大值更會使誤差最大化，導致過度估計(overestimate)的發生。Double DQN 就是為了改善此問題而被提出的，式(8)為新的目標 Q 值計算方式，此算法避免掉式(2)造成的過度估計問題，使網路更新變得更加穩定：

$$Q_{target} = R_t + \tau \times Net_{target}(S_{t+1})\left[argmax\left(Net_{main}(S_{t+1})\right)\right] \qquad (8)$$

3. Dueling DQN

一般 DQN 會直接輸出每個動作在目前狀態所具有的價值，但 Dueling DQN[5]會先拆分出狀態隱含的價值 V(純量)與動作具有的優勢 Advantage(向量)，再將其相加才會得出所求的價值 Q：

$$\mathrm{Qs} = \mathrm{V} + \mathrm{Advantage} \qquad (9)$$

然而，如此計算無法從給定的 Q 值反推出唯一的 V 與 A，這會導致網路的性能嚴重降低，因此會將 A 減去最大值後才與 V 相加：

$$\mathrm{Qs} = \mathrm{V} + (\mathrm{Advantage} - \max(\mathrm{Advantage})) \qquad (10)$$

但是實務上使用時，為了提高模型穩定度，會把最大值改為平均值：

$$\mathrm{Qs} = \mathrm{V} + (\mathrm{Advantage} - \mathrm{mean}(\mathrm{Advantage})) \qquad (11)$$

3.3 自我訓練

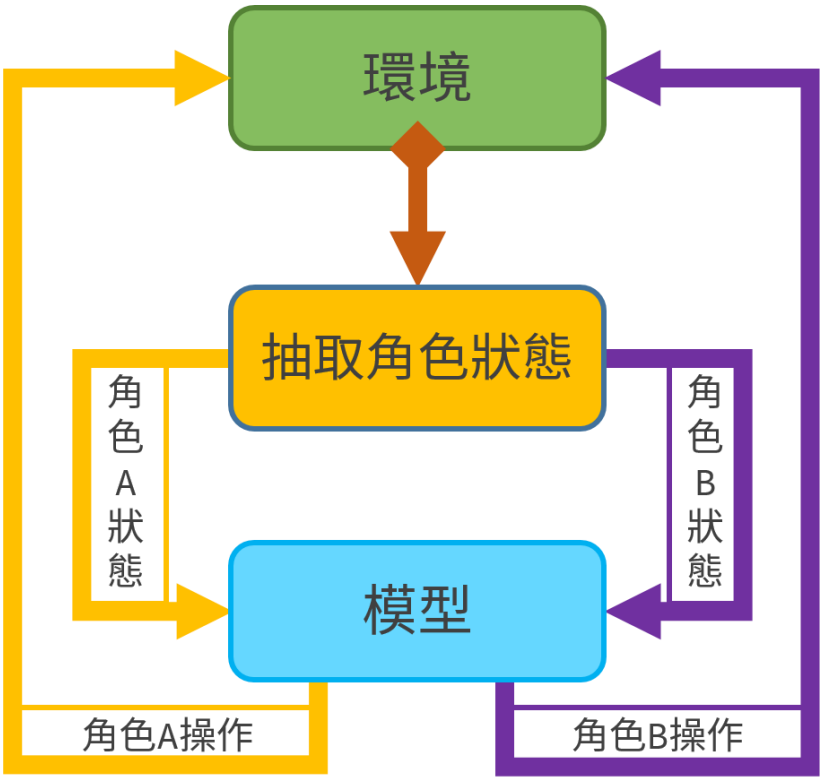

圖 3 自我對戰示意圖

為降低人力成本，訓練所需的對戰資料會由模型進行自我對戰產生(如圖 3)。然而格鬥遊戲的特性是由兩個具備相等條件的角色進行對戰，若使用圖像作為輸入，只使用單一模型時會無法理解哪一個是目前在操作的角色，因此本研究使用角色的狀態向量替代圖片作為輸入，這樣就能簡單的區別操作角色。且由單一模型操作雙角色亦可增加訓練所使用的資料量，以加快學習進步速度。

3.4 超參數測試

模型的效果除了與算法有關外，還會被包含像是類神經網路的層數、寬度，以及DQN[3]中的折扣係數、獎勵函數等等超參數(Hyperparameters)影響，需要人為調整、測試才知道好壞。因此在訓練完成後若得出結果不甚理想便需要再度調整超參數並重新訓練。

四、系統架構

4.1 運作流程

本專題的系統流程如圖 4 所示，當啟動遊戲環境後，網路模型與環境每幀的互動資料就會被收錄進記憶池中，當一回合對戰結束後就會抽取出記憶池內的經驗開始訓練並優化模型。當結束一回的訓練後就會開始新對戰，藉由重複對戰、訓練的循環使網路學習更佳的操作技巧。

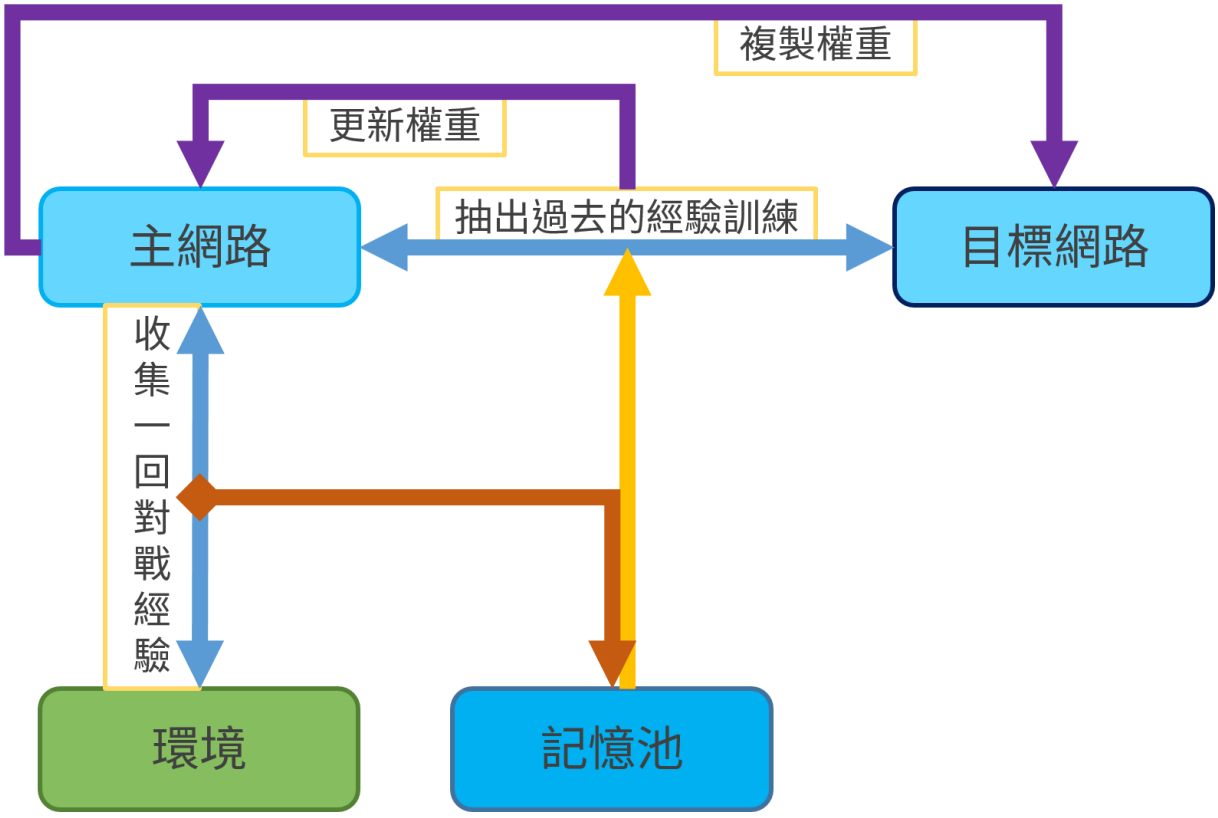

圖 4 系統流程

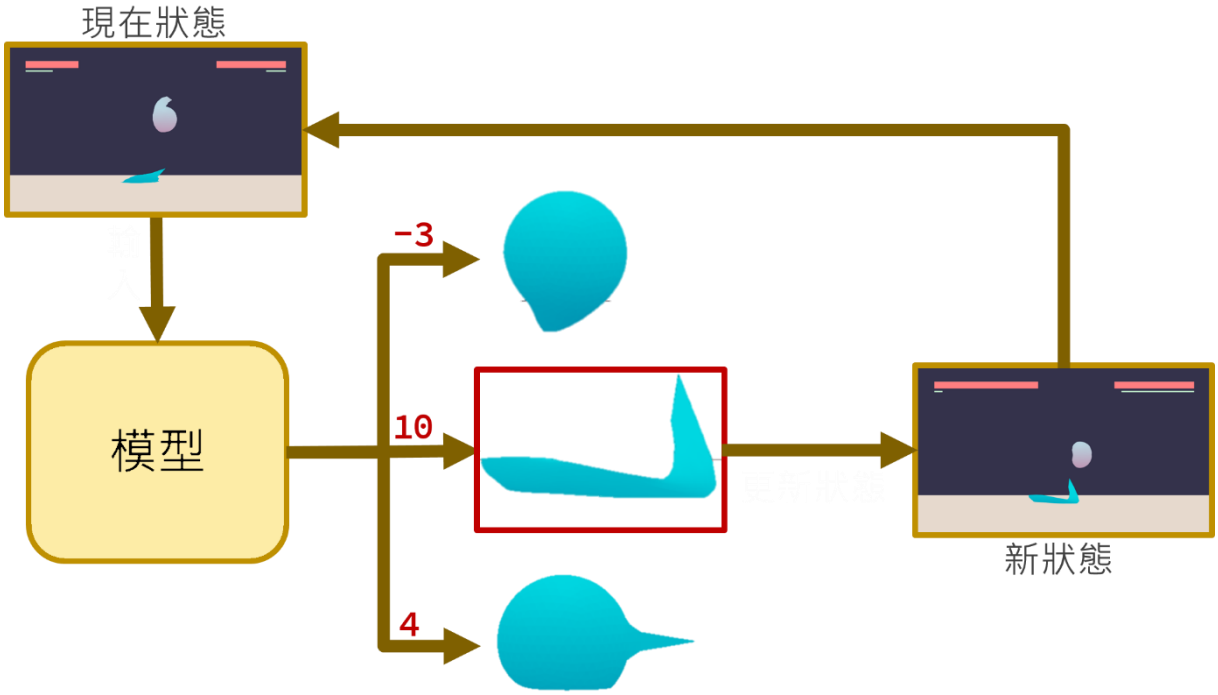

圖 5 決策流程

實際對戰的運行方式如圖 5 所示，當模型估測出所有動作的價值後，就可藉由 Epsilon Greedy 挑選動作，Epsilon Greedy 挑選的作法是有ε的機率隨機選擇動作，剩下 1 − ε就挑選有最高價值的動作。在學習階段時可以調高ε來增加探索性，讓模型更快的學習到良好的操作方式，而在實際應用時降低ε就能使模型穩定發揮所學會之技巧。

4.2 輸入與輸出格式

輸入由[我方狀態向量,我方上一個操作,敵方狀態向量]串接而成，而狀態向量則是由血量、負荷值、位置、跳躍次數、面對方向與目前的動作(與操作不同)等等資訊組合而成。

輸出是<上/下/左/右/輕攻擊/中攻擊/重攻擊>這七個按鍵按與不按的組合所能獲得的分數，共計有$2^7 = 128$個輸出。

五、實驗成果與討論

5.1 實驗環境

表 2 環境配置

處理器	Intel® Core™ i5-4570 CPU @3.20GHz
記憶體	8.00GB
圖形處理器	NVIDIA Geforce GTX 1060 6GB
作業系統	Windows 10 64 位元作業系統、Linux Mint 64 位元作業系統
開發環境	Chrome 77 版以上、Node.js
使用語言	JavaScript
函式庫	Tensorflow.js、Babylon.js

表 2 為本專題的環境配置。運用 JavaScript 跨平台的優勢使本專題能於各作業系統上快速建置開發環境，此外 Tensorflow.js 可利用 webGL API 調用 GPU 的平行運算能力加速計算速，並且不必像 Tensorflow.py 還需要另外安裝 CUDA。

5.2 成果與討論

實驗過程中，如果隨機挑選經驗池中的記憶進行訓練，會導致學習速度緩慢。因此，在實作時增加了簡易的優先經驗重播機制，計算出每筆記憶的預測誤差，讓誤差大經驗的有更高的機率被抽出來變成訓練資料，大大的加快了進步速度。

關於獎勵函數的設定，在經過多次研究後發現，若獎勵函數設計的過於複雜，將會使模型的學習效率低下。為此，將獎勵函數設定為：

if 被敵人擊中：
　負值獎勵(大)
else if 擊中敵人：
　正值獎勵(大)
else：
　我方血量百分比-對方血量百分比

在測試過程中，嘗試將激活函數從 selu[6]換成 mish[7]後，模型的應對能力又更加優異。通過實驗調整過各種超參數的最終版本模型，在歷經 96 小時的自我訓練之後已獲得不錯的應對能力，面對將人類作為對手的場景也不會落入下風，在經過多人測試後得到超過 7 成的勝率。

六、結論

本專題降低了設計格鬥遊戲 AI 的複雜度，只需要調整獎勵函數就可改變 AI 的行動模式，且訓練出來的模型能於對戰中取得不錯的勝率。但此研究最終目的是製造出與玩家一同成長的勁敵 AI，因此，未來將測試不同強化學習方法與各類模型搭配組合後的效果，讓模型學會「放水」，藉由這種方式模擬出「勁敵」，並藉由與勁敵競爭所帶來的刺激感來提高玩家對遊戲的依賴度。

除此之外，本專題目前是將角色的各種狀態作為輸入使用，但這樣會出現以下兩個缺點：

1. 要去額外設計輸入格式，就算經過設計也不一定能正確表達角色的狀態向量。
2. 模型的通用性低下，一個模型只能學習一種對戰組合，如表 3 所示當有 N 個角色就需要N^2個模型。

表 3 角色配對示意

對手 / 我方	**角色 A**	**角色 B**	**角色 C**
角色 A	模型<A,A>	模型<A,B>	模型<A,C>
角色 B	模型<B,A>	模型<B,B>	模型<B,C>
角色 C	模型<C,A>	模型<C,B>	模型<C,C>

因此未來的改進方案希望能結合 word embedding 的方式，讓模型自行學習編碼狀態向量，並如圖 6 所示，將模型拆分成角色狀態編碼、環境映射與角色動作映射三區塊，如此就只需要準備與角色數量相等的<狀態編碼>與<動作映射>，這不僅減少模型的數量還達成更高的通用性。

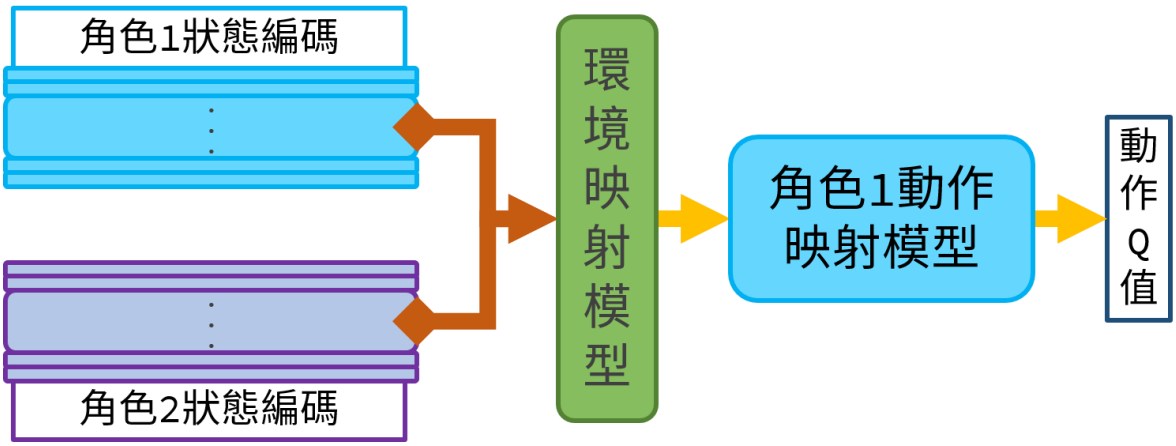

圖 6 未來架構

參考文獻

[1] CS 294 Deep Reinforcement Learning (UC Berkeley), http://rail.eecs.berkeley.edu/deeprlcourse-fa17/, 2017.

[2] Tensorflow.js, https://www.tensorflow.org/js/, 2019.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, “Playing Atari with Deep Reinforcement Learning,” arXiv preprint arXiv:1312.5602, 2013.

[4] Hado van Hasselt, Arthur Guez, David Silver, “Deep Reinforcement Learning with Double Q-learning,” arXiv preprint arXiv:1509.06461, 2015.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,”arXiv preprint arXiv:1511.06581, 2015.

[6] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter, “Self-Normalizing Neural Networks,” arXiv preprint arXiv: 1706.02515, 2017.

[7] Diganta Misra, “Mish: A Self Regularized Non-Monotonic Neural Activation Function,” arXiv preprint arXiv:1908.08681, 2019.

附錄一、Double DQN 運算圖解

Double DQN 的推導較為複雜，因此於附錄中配合圖解方式拆解分述其運算步驟：

1. 由Net_{main}利用狀態S_{t+1}計算出Qs_{t+1}，並選擇有最大值的動作成為A_{next}：

$$A_{next} = argmax\left(Net_{main}(S_{t+1})\right) \tag{12}$$

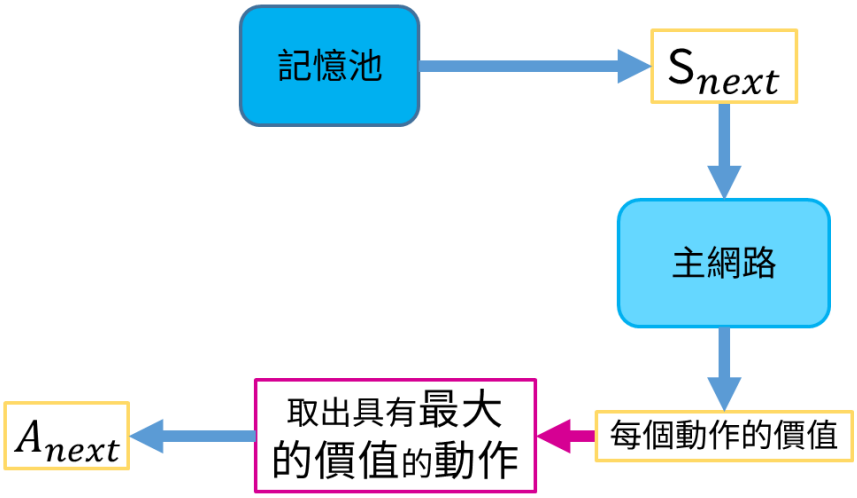

圖 7 公式(12)

2. 由Net_{target}利用狀態S_{t+1}計算出Qs'_{t+1}，並選擇動作A_{next}的 Q 值計算Q_{target}：

$$Q_{target} = R_t + \tau \times Net_{target}(S_{t+1})[A_{next}] \tag{13}$$

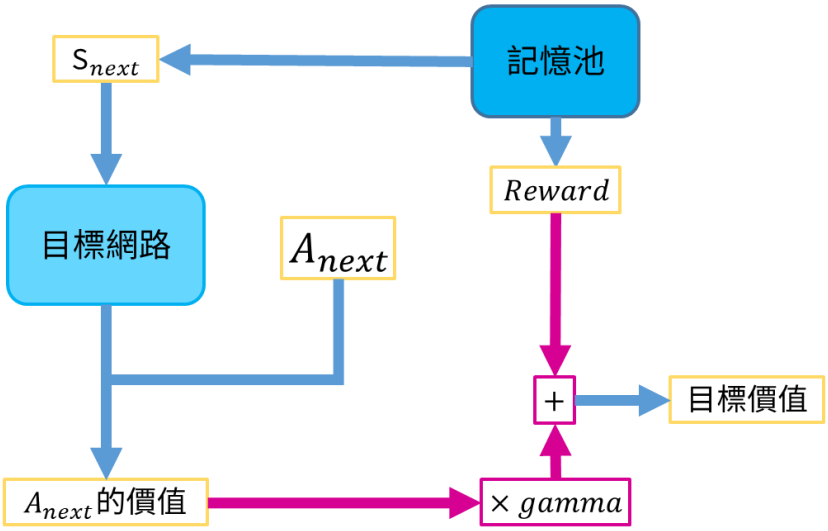

圖 8 公式(13)

3. 由Net_{main}利用狀態S_t計算出Qs，並選擇記憶池中的動作 A 得出Q_{main}：

$$Q_{main} = Net_{main}(S_t)[A_t] \quad (14)$$

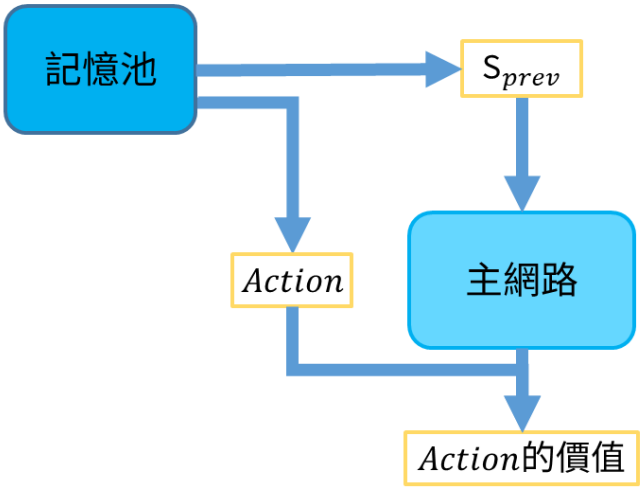

圖 9 公式(14)

4. 利用 mean square error 計算出 loss 後就可對Net_{main}的權重進行更新，而在更新數次後再將其權重複製給Net_{target}：

mean square error(Q_{main}, Q_{target})更新Net_{main}的權重 (15)

Net_{main}更新數次後將權重複製給Net_{target}

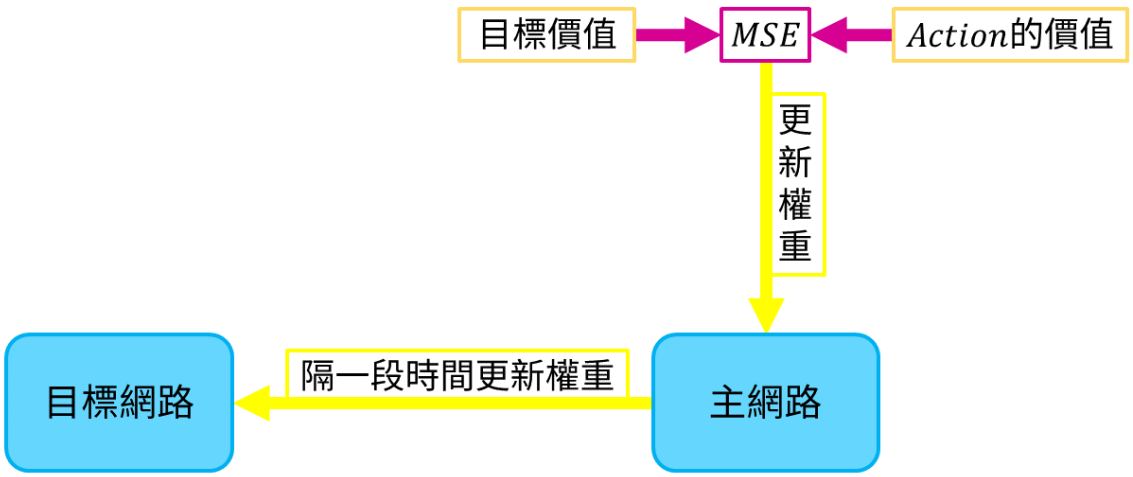

圖 10 公式(15)

附錄二、環境建置

本專題已放置於 github 上，主程式放置在 dddqn<i>中，而模型則位於 src/js/MirageNet/dddqn<i>中，安裝流程如下所示：

1. 安裝 node.js 與 npm：
   「https://nodejs.org/en/」

2. 安裝 git：
   「https://git-scm.com/」

3. 將 https://github.com/toonnyy8/AI-of-FTG.git clone 至本地：
   執行「git clone https://github.com/toonnyy8/AI-of-FTG.git」

4. 進入 AI-of-FTG 資料夾。

5. 安裝依賴庫：
   執行「npm i」

6. 啟動開發環境：
   執行「npm run dev:dddqn<i>」後開啟「127.0.0.1:8080/index.html」與「127.0.0.1:8080/agent.html」
   打包環境：
   執行「npm run build:dddqn<i>」
   將<i>換成想要執行的版本，詳細版本執行指令可查看 package.json。

附錄三、軟體手冊

(A) AI of FTG 設定介紹

當開啟 AI of FTG 後會有各式功能設定項目，每個設定項目的代表意義如下：

1. **Control period：P**
   用於調整模型的操作週期，代表 P/60 秒會發出一次控制訊號。
2. **Player AI**
   是否開啟 AI 控制代理。
3. **Player AI choose action random value : ε**
   會依照 Q 值大小隨機選擇動作的機率。
4. **Correction probability**
   啟動機率補正，使機率平均化。
5. **Automatically reduce HP**
   自動降低血量，可避免自動操作一直不讓對戰結束。
6. **Train at every frame：N**
   每 N 幀後就進行訓練。
7. **Train at every end**
   每場戰鬥結束就進行訓練。

(B) 遊戲操作

AI of FTG 使用自製的格鬥遊戲 silme FTG 做為測試環境，以下將會介紹 silme FTG 的遊戲系統與操作方式：

1. 過載系統

i. 當受攻擊時會增加過載計量表的數值，增加量等同受到的傷害。
ii. 當使用同樣攻擊時，會增加過載值。
iii. 過載計量表在未填滿的狀態，會自動減少過載值。
iv. 當過載計量表全滿時又受到攻擊，會額外追加等同於過載計量表的傷害(不受連擊補正影響)，並將計量表歸零。
v. 當過載計量表全滿時將無法取消受擊硬直。
vi. 取消受擊硬直可減少過載計量表(詳閱「取消受擊硬直」)。
vii. 完美防禦可減少過載計量表(詳閱「防禦系統」)。
viii. 當過載計量表全滿時，強攻擊將不帶有霸體狀態(詳閱「攻擊系統」)。

2. 攻擊系統

i. j/k/l 與 1/2/3 為輕/中/重攻擊。
ii. 重攻擊實行時會有霸體效果(攻擊不會被中斷) 。
iii. 連擊補正：當產生連擊時，打出的傷害會依照連擊數下降。

3. 防禦系統

i. 當受攻擊時移動方向為向後(遠離對手)，將會觸發防禦，此時受到傷害與過載值變為原本的 1/2。
ii. 當受到攻擊前 6 幀(約 0.1 秒)往後移動，將觸發完美防禦，此時將完全擋下傷害，並返回過載計量表給對手。

4. 取消受擊硬直

在過載計量表未全滿時，於受擊硬直回復的過程種跳躍就可取消硬直，並將過載計量表歸零，但會受到「現有過載值」*「血量百分比」的反噬傷害。

圖 11 為成品的展示圖，在畫面靠上較粗的方條(紅色)是血量條，而位於血量下的細方條(白色)則是過載計量表，當過載計量表全滿時會陷入不利的狀態，但同時也是逆轉的契機。

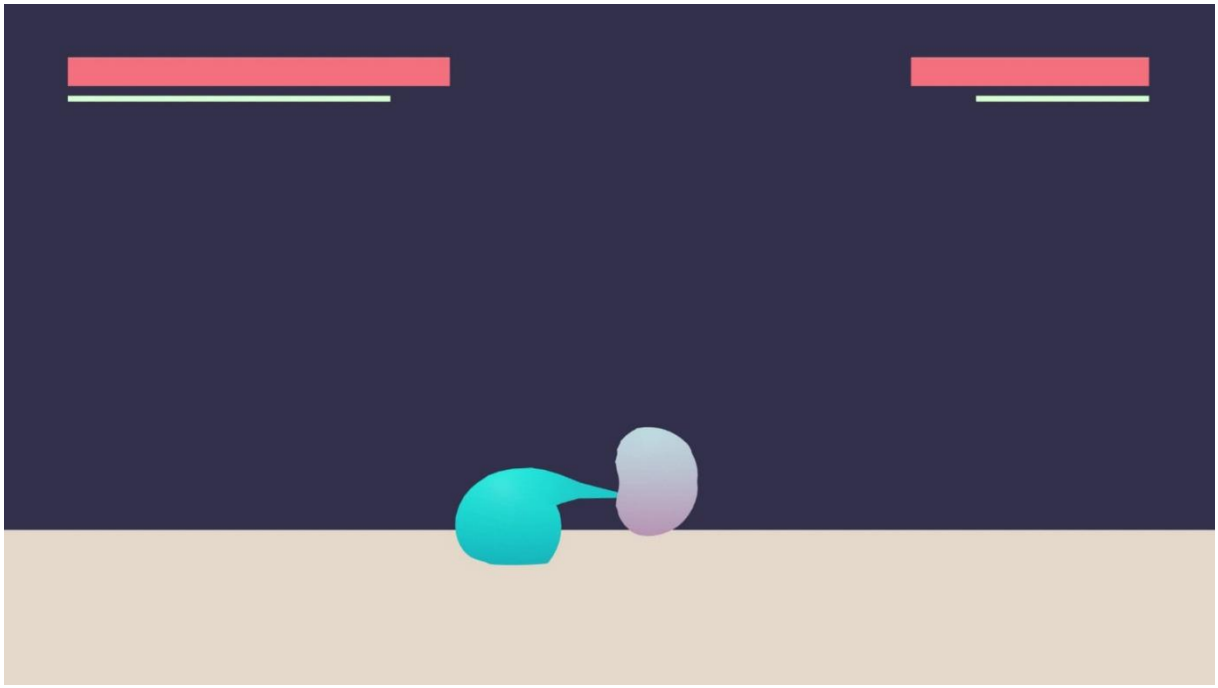

圖 11 成果展示圖

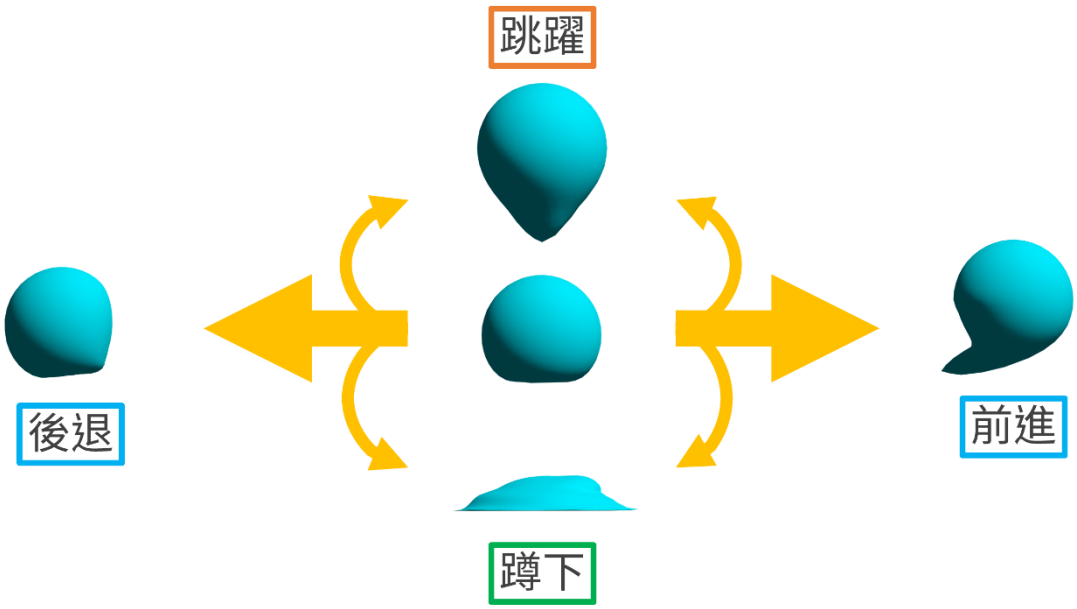

圖 12 移動

當按下<上/下/前/後>時就會做出相對應的移動動作，若跳躍時再度跳躍可發動二段跳，且二段跳時可配合<前/後>改變移動方向，而當蹲下時是無法移動的。

攻擊有分為<輕/中/重>三種不同的攻擊，每種攻擊各有自己的特點，其中重攻擊發動時是無法被打斷的。

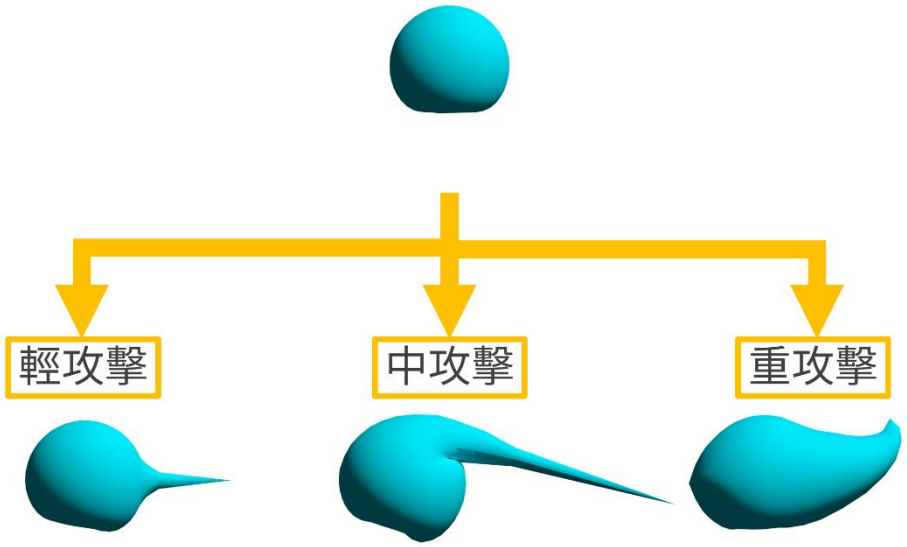

圖 13 攻擊

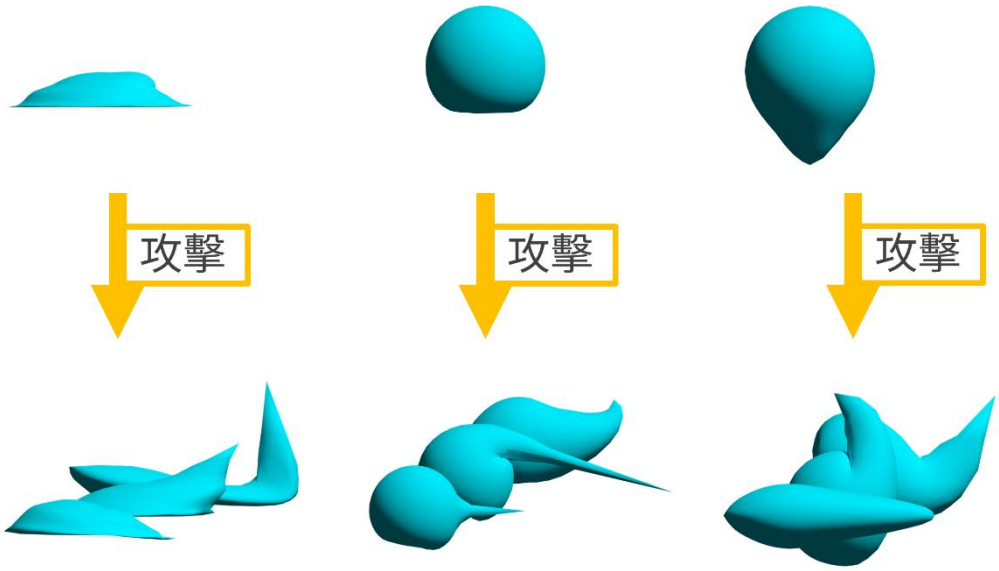

圖 14 蹲下、站立與跳躍各有不同攻擊

當被攻擊時有按下後退就會自動發動防禦使傷害降至 1/2，就算在跳躍或蹲下時亦會發動，當在受攻擊的 10 幀內按下防禦，就可發動完美防禦，這時就能完美抵擋傷害且不會有硬值，而自身過載計量表也會完全疊加到對手上。

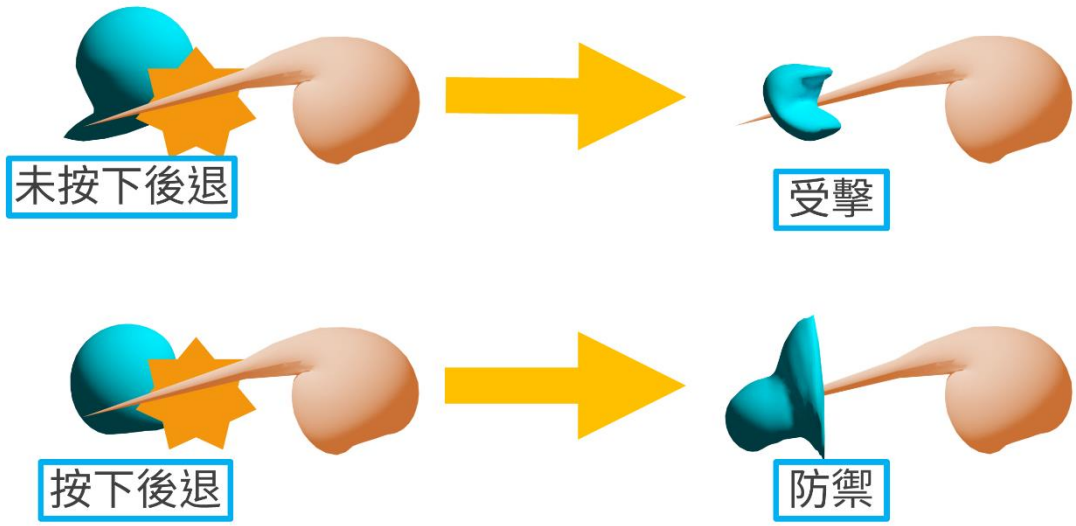

圖 15 防禦

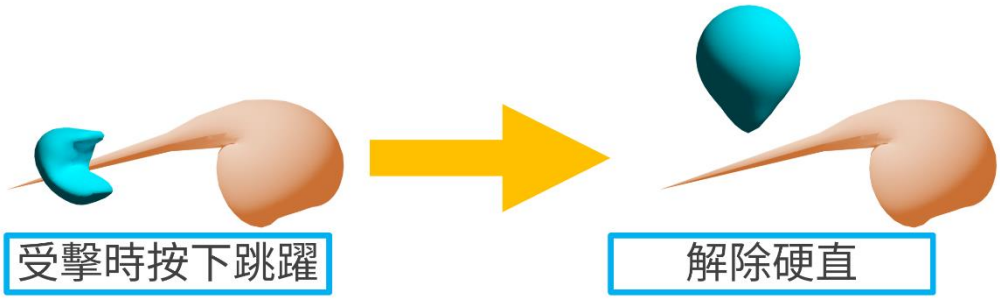

圖 16 取消硬直

被攻擊打中時會使過載計量表增加並出現硬值，但此時只要按下跳躍就可取消硬值，並讓過載計量表歸零，但出現傷害反噬。此外，當過載計量表全滿時便無法取消硬值。

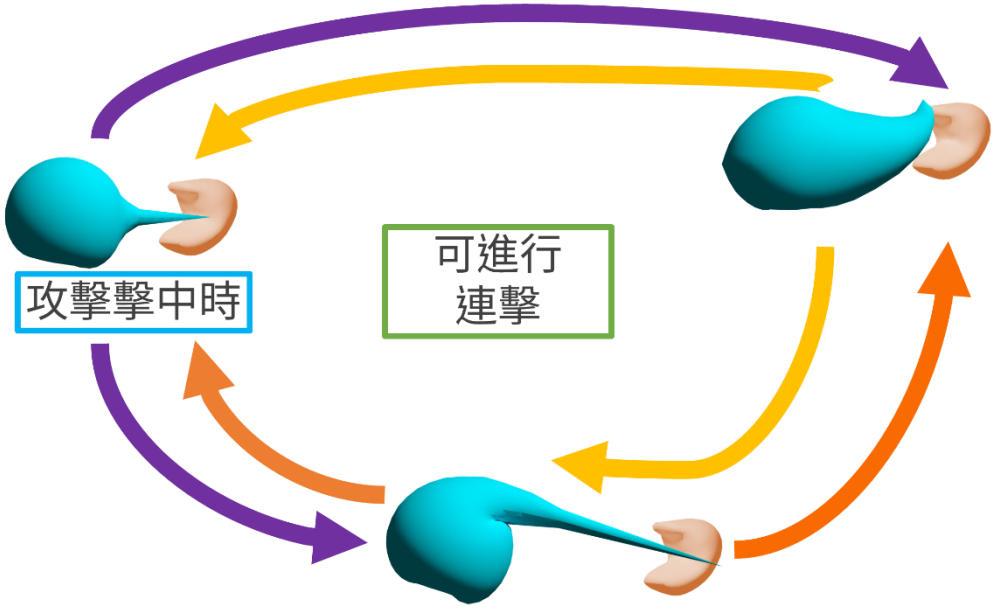

圖 17 攻擊派生

當攻擊成功擊中對手時，使用同一個攻擊會出現硬值，但若是銜接其他攻擊就不會出現破綻進而達成連擊。