HackMD link (https://hackmd.io/@libao3128/S1tc3yGnO)
Team member:黃立鈞、簡言哲、卓灝辰、廖振亦

Introduction to the problem

利用NEAT(NeuroEvolution of Augmenting Topologies)演算法讓AI操控經典電腦遊戲「小朋友下樓梯」。
遊戲中，玩家可以操控左右方向鍵讓小朋友向左或向右移動，隨著平台不斷上升，目標是讓小朋友能夠往下最多的樓層數。
小朋友一共有10點生命值，在接觸到天花板的尖刺或是尖刺平台後都會受到4點傷害，當小朋友被天花板的尖刺傷害時會強迫離開所在的平台往下跳，而當生命值歸零或是小朋友掉落谷底時遊戲結束。
平台的種類分為:

1. 普通平台:
   沒有任何效果，每當小朋友站上此平台可以回復一點生命值。
   

2. 假平台:
   假平台會在小朋友踩過後翻轉，並在翻轉期間不能作為平台使用，每當小朋友站上此平台可以回復一點生命值。
   

3. 彈簧平台:
   彈簧平台會在小朋友踩過後將其往上彈一段距離，可以重複並且同時使用，小朋友不能藉此平台回復生命。
   

4. 履帶平台(向左或向右)
   小朋友在踩上履帶平台後會因為履帶平台往右或往左轉動而向左或向右平移，但可以透過自身位移抵銷，每當小朋友站上此平台可以回復一點生命值。
   
   向右履帶
   
   向左履帶

5. 尖刺平台
   在小朋友踏上平台後給予他4點傷害。
   

GitHub code we refer to

1. NEAT-Python (https://github.com/CodeReclaimers/neat-python/blob/master/docs/index.rst)
   這次project中，我們使用NEAT-Python套件作為我們神經網路的建構模組，透過修改模組中的config檔，我們可以輕易地修改整個網路的許多有關Reproduction、Genome、Stagnation的參數，相關的code為config-feedforward檔。
2. visualize.py (https://github.com/CodeReclaimers/neat-python/blob/master/examples/xor/visualize.py?fbclid=IwAR2UfJ_GkPcvN0Hen4RpTXbQbcZD_YSSh7mSYZ66Z1cAsatyfJbUycOnkLg)
   visualize是NEAT_Python中包含將模型視覺化的檔案，需要另外安裝Graphviz (https://graphviz.org/)套件才能使用，方便我們了解現在模型的node、connection、weight等相關資訊。

Algorithm

Code

1. GameObject:
定義遊戲中的物件
(1). buildground:
建立基本的背景，包括背景畫布(800x600，左上角是(0,0))，以及牆壁，天花板，目前狀態(目前層數，小朋友數量，這次train中所抵達過的最低樓層)這些不會移動的物件在背景上的位置。
(2). 各平台:
各個平台都有三個function，一個是initial，一個是generate，一個是update。initial定義一個平台的基本數值(x座標，y座標，平台類型等);generate定義平台產生的方式，如果是本身會有動畫的平台在這邊需要切換不同圖檔形成動畫的效果;最後是update，是用來確認下一個平台應該有的狀態，若是碰到天花板就return flase，否則往上平移。
(3). player:
定義小朋友的function有四個，比起平台多了ClosestPlatform，這個function的目的是要找出比小朋友還低並且離小朋友最近的兩個平台;小朋友的initial跟generate都跟平台的相去不遠，但update裡的東西就多了些，小朋友的update要確定的狀態有血條，座標，以及生死。首先是血條，要先確定是否碰到天花板或尖刺平台，然後計算血條再給血條的長度以及顏色。生死的部分是如果血條歸零就切到死掉的圖檔並落下到底並消失。座標要先確定移動的方式，向左向右或著不動，如果碰到牆壁就x座標不動，碰到平台則y座標不動，沒有在平台上便以設好的速度下墜。

2. AI_FinalProject:
(1). play:
與name的功能(1)連結，是拿來玩小朋友下樓梯的，功能有生成小朋友還有平台，還有對這兩者進行更新(update)
(2). PrintP1State:
把隨機一個小朋友的給config的input印出來(但會卡所以先不顯示)
(3). evaluate:

計算fitness(小朋友死掉時的樓層數 * 10+每50個操作循環時小朋友的生命/10-被天花板或是平台的尖刺刺到的次數x3)，還有把config傳回來的output反映到小朋友下一個行為上，隨機生成平台，更新抵達過的最低樓層(max_floor)。

(4). run:

將evaluate需要的config、genome參數先處理好，同時增加reporter、checkpoint方便我們在訓練過程中記錄與監測，最後儲存run返回的最佳model且將model視覺化。

(5). test:

用來測試過去訓練好的model在實際的應用情形。

(6). Load_fit:

與run類似，但這裡是使用過去訓練的checkpoint繼續fit。

(7). name:

執行程式一開始的主畫面，可以執行不同功能，

1. 是玩遊戲，可以讓使用者透過空白鍵生成並用方向鍵控制小朋友。
2. 利用設定好的config檔與input、output重新開始訓練，預設是100個generation。
3. 是載入過去生成小朋友並測試他的表現，結束可以透過輸入1重新測試。
4. 是用輸入的checkpoint中的module繼續做training與還要training幾個generation。
   在訓練的過程中可以從輸出看到每一個generation生成的情形、現在有的species、與model的大致狀況。

3. visualize:

為參考NEAT-Python上的code，用來呈現我們訓練好的model的樣貌與訓練的結果。

4. config-forward:

是NEAT model中最重要的設定檔，調節NEAT生成中的一切參數，其中分為4個section。

NEAT

控制有關每一輪訓練的參數
fitness_criterion:max，意思是我們在評量每一個species時要用整個群體的最大值作為比較。
fitness_threshold:2000，當fit_score大於2000時，我們認為模型訓練完成，則停止訓練以避免overfit。
pop_size:50，每一輪的訓練中生成50個小朋友。

DefaultGenome

activation_default=tanh，使用tanh作為activation_function。
conn_add_prob=0.5，每一輪有50%的機率生成新的connection。
conn_delete_prob=0.5，每一輪有50%的機率刪除既有的connection。
initial_connection=full，在一開始採用fully connection連接所有的input與output。
num_hidden=0，一開始有0個hidden layer。
num_inputs=12，有12個input。
num_outputs=2，有2個output。

DefaultStagnation

max_stagnation=20，當一species在20次生成後沒有進步，他就會被淘汰。
species_elitism=5，有5個species會被保護，以免被淘汰。

DefaultReproduction

elitism=2，所有species裡最好的兩個會被保留到下一個generation。
其他還有許多參數可以設定，因為篇幅關係不一一列舉，沒有特別提到的大多是參考網路上其他project的設定方式。

Alogorithm

再NEAT的訓練過程中，每一輪會產生50名小朋友(pop_size)，在一開始，小朋友會根據他的input做出隨機行為(output)，所有小朋友會根據他蒐集到的input與fit_score演化，並生成能夠獲得更高fit_score的神經網路。

input

我們的input有12種，分別是:

1. x:小朋友的位置的x座標。
2. y:小朋友的位置的y座標。
3. on_platform:小朋友是否在平台上，若是則輸出1，否則輸出0。
4. life:小朋友所剩的生命值(0~10)。
5. closeXL:小朋友的x座標-離小朋友最近的平台其左緣的x座標。
6. closeXR:小朋友的x座標-離小朋友最近的平台其右緣的x座標。
7. closeY:小朋友的y座標-離小朋友最近的平台的y座標。
8. ToRight:場地左緣的x座標-小朋友的x座標。
9. ToLeft:場地右緣的x座標-小朋友的x座標。
10. IsTram:離小朋友最近的平台是否為彈簧平台，是則輸出1，否則輸出0。
11. IsNail:離小朋友最近的平台是否為尖刺平台，是則輸出1，否則輸出0。
12. IsLR:離小朋友最近的平台是否為左或右履帶平台，是右履帶平台輸出1，是左履帶平台輸出-1，否則輸出0。

其中，離小朋友最近的平台的定義為:所有位置在小朋友之下的平台中，y座標與小朋友相差最少的，也就是小朋友下一個最有機會跳上的平台，在遊戲畫面中的黃色圈圈就是各個小朋友所看到的最近平台。

output

我們的output有兩個，分別是:

1. GoLeft:當output大於0.5時，則控制小朋友往左走。
2. GoRight:當output大於0.5時，則控制小朋友往右走。

若兩個output同時大於0.5時，則GoRight優先，當兩個output都小於0.5時，則小朋友會停在原地。

fit_function

fit_function是用來計算小朋友fit_score的函數，是模型用來衡量小朋友表現的重要指標，因此會直接影響到模型生成的走勢。

我們定義的fit_function為:
fit_score=
+小朋友死掉時的樓層數 * 10
+每50個操作循環時小朋友的生命/10
-被天花板或是平台的尖刺刺到的次數*3

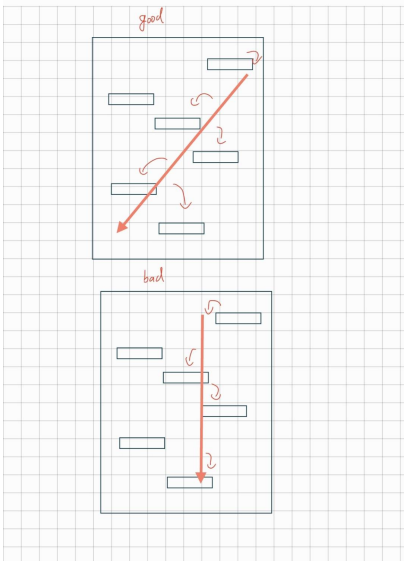
目的是讓小朋友盡可能地往下更多樓層，並保持高的生命值與減少被尖刺傷害的次數。

Result

在經過我們反覆調教不同的參數設定與長時間的generation後，我們的小朋友成功達到60階的最高紀錄，而附檔中提供的winner是曾經在訓練中到達過55層並且在單獨測試50次中達到最高42層的model，可以透過程式中第3個選項來重現他的表現，而對於這個結果我們認為還有進步的空間。

策略

在訓練的過程中，我們發現小朋友會發展出特定的行為模式，在不同的參數設定下會有些許不同，但整體來說我們可以發現，大多數拿到高分的小朋友會等到所在的平台上升到快到天花板時才向左或向右移動離開平台並向下反方向移動，我們推測這是因為在小朋友的input中並沒有辦法看到整個場地中所有平台的位置與類型，唯一能看到的只有自己腳下的，當他隨著平台往上升時，他會從平台的右邊往下跳並向左移動，如此一來他有更大的機率能夠跳到下一個平台上，如圖所示。



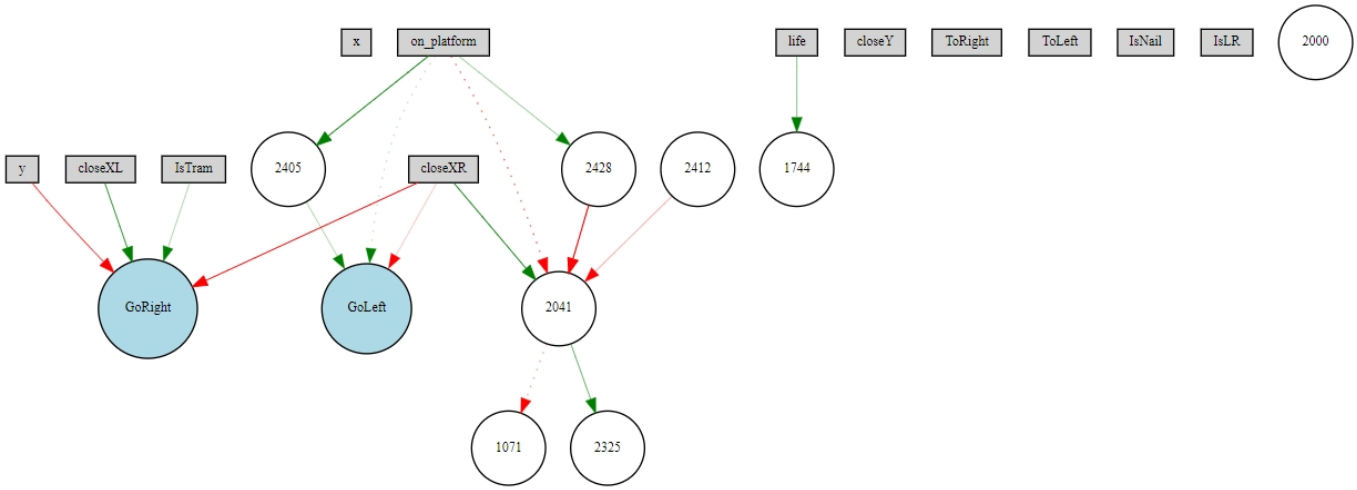
然而，這個策略最大的缺點就是當小朋友所在的平台位置太過右邊時，小朋友會被牆壁卡住而無

法往下跳，如此一來便無可避免被天花板的尖刺傷害且強迫向下跳。而當小朋友所在平台為彈簧平台且位置靠右時，小朋友既無法向右往下跳也無法透過天花板的尖刺強制向下跳，就容易被彈死在天花板跟彈簧平台中間。

以下是我們附檔提供的pretrained model的model diagram，我們可以發現，其中有5個在generation中生成的hidden layer，而且有約一半的input在這個模型中沒有跟其他node有connection，但我們任為這不代表這些input是沒用的，只是在這個模型中剛好被刪除，在其他模型中是有可能有連接的。connection中，箭頭的粗細代表著connection的weight的大小，而顏色代表的是正權重或是負權重，虛線則代表那個connection在那一個generation是被disabled的。

我們可以清楚的看到不同的input對於output有什麼影響，例如closeXR對於GoRight就有很大的影響。

我們也可以看到y對於GORight的負相關，代表當y座標下降(小朋友上升)時會傾向往右走，這跟我們觀察到的結果相符。



對於不同平台的應對

我們發現，當小朋友位在不同種類的平台時會有相對應的策略:

- 履帶平台:在履帶平台上，小朋友會為了不要提早落下，而在履帶的邊緣輸出相反的動作，例如:在向右的履帶平台上，小朋友會向左走以防自己掉落，如此一來小朋友可以卡在履帶的邊緣等到他想往下跳時才往下跳。
- 彈簧平台:彈簧平台可謂小朋友的殺手，因為彈簧平台會不斷的讓小朋友往上彈而讓小朋友以為自己不在平台上而不斷的去找最近的平台著落也就是自己所在的彈簧平台，當小朋友想遵照自己的策略往右向下找平台時他會誤以為自己已經向下跳了於是開始早最近的平台，如此一來便很難離開彈簧平台。但隨著時間的演化，有小朋友發現這個情形便會在彈簧平台一次一次的反彈中慢慢向右移動借此離開平台，但仍有很大的機率在反彈時被天花板刺死或卡在天花板與平台中間。
- 尖刺平台:在少數的參數下，有些小朋友發現自己會因為尖刺平台而受傷進而影響fit_score，於是在下落時選擇不跳在尖刺平台上，但經常會因此沒有落腳處而死亡，因此大部分的策略下，小朋友還是會選擇落在尖刺平台上以減少摔死的風險。
- 靠右的彈簧平台+天花板:靠右的彈簧平台與天花板的組合是AI小朋友的最大剋星，因為小朋友脫離彈簧平台的方式是慢慢向右彈並在邊緣處落下，但靠右的彈簧平台無法在右邊落下同

時也因為彈簧平台的特性無法透過被天花被傷害而強制落下，因此當小朋友遇到這個組合有很大的機率死亡。

總結

在歷經上萬次的generation後，我們得出以下結論:

1. **小朋友的表現與他的策略並沒有絕對關係**，平台生成的類型與分布更會影響當輪小朋友的表現，在某些情況下，有特定策略的小朋友表現甚至會比從頭到尾待在原地的小朋友差，因為策略反覆移動小朋友的關係，小朋友要冒更大的風險去找下個平台，有不成功便成仁的感覺。
2. **同樣的參數下升成的模型與策略截然不同**，因為在NEAT的模型中，本身就帶有許多隨機的參數以增加小朋友變異的可能性，因此在同樣的設定下如果運氣不好就無法生成良好的模型，這也意味著良好的模型很難被複製，只能透過chekpoint把好的模型留下來。
3. **越長時間的generation並不代表更好的模型**，就如同其他的模型一樣，過久的generation或fit有可能導致overfit，尤其在NEAT模型中，node之間的connection是有可能被切斷的，當重要的connection被切斷時小朋友的表現可能一落千丈。
4. **小朋友找bug**，我們原本遊戲的設定中沒有限制小朋友只能連續藉同一個平台回一次血，因為我們預設小朋友沒有辦法著陸一個平台兩次，但在generation的過程中，我們發現AI找到了一個方式能夠在普通平台與履帶平台邊緣反覆地離開與著陸，藉此回復大量生命值，後來我們便把這個bug修正了，從此再也沒有作弊的小朋友。
5. **超過一半的小朋友會死在遊戲剛開始時**，在一開始的時候，小朋友會從場地的正中間落下，有超過一半的小朋友會因為平台數量或生成過多尖刺和假平台導致在遊戲一開始就死亡，在某些generation中，有些策略能讓小朋友平均有約8成的機率可以在遊戲開始成功著陸，但這些策略在之後的表現大都不好。
6. **最後留下來的species不一定是最好的**，在訓練的過程中，有發生過一些訓練良好、有特定策略的species被淘汰，原因有可能是因為連續生成的平台不適合他的策略，導致在多個generation後沒有進步或是甚至比不過呆在原地的小朋友，導致最後被淘汰，但通常這種有特定策略的小朋友才是我們想要訓練出來的，這時就需要我們手動終止訓練，並用checkpoint回到他還沒被淘汰的時候，多給他幾次機會。