EECS207001: Logic Design Laboratory

Final Project:

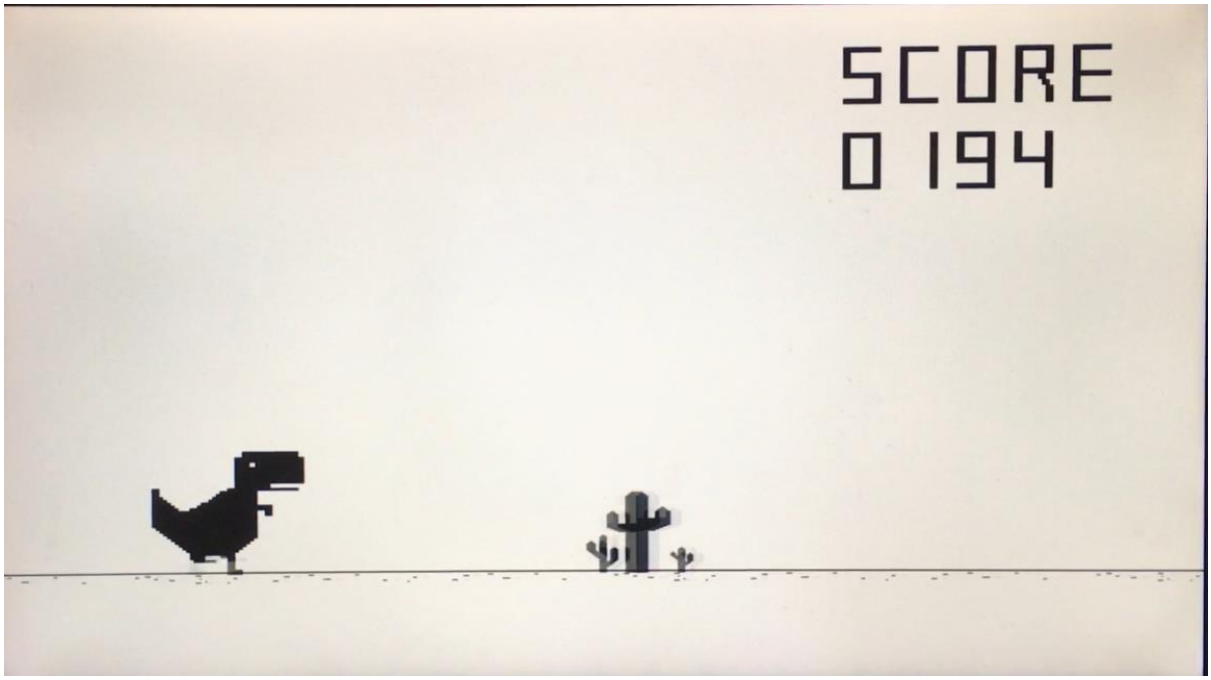
Dino on FPGA

report

侯皓予、阮柏諭

2022/1/14

Introduction



這是一個 chrome 的小恐龍遊戲，可以在兩種模式中切換，第一種是普通遊玩，第二種是開始 Q learning。用鍵盤操控，按空白鍵開始遊戲，方向鍵上或空白鍵是跳。按下鍵盤 Q，開始自動 Q learning，此時玩家不需輸入，僅看他學習的過程。FPGA 中間那個 button 是 reset。

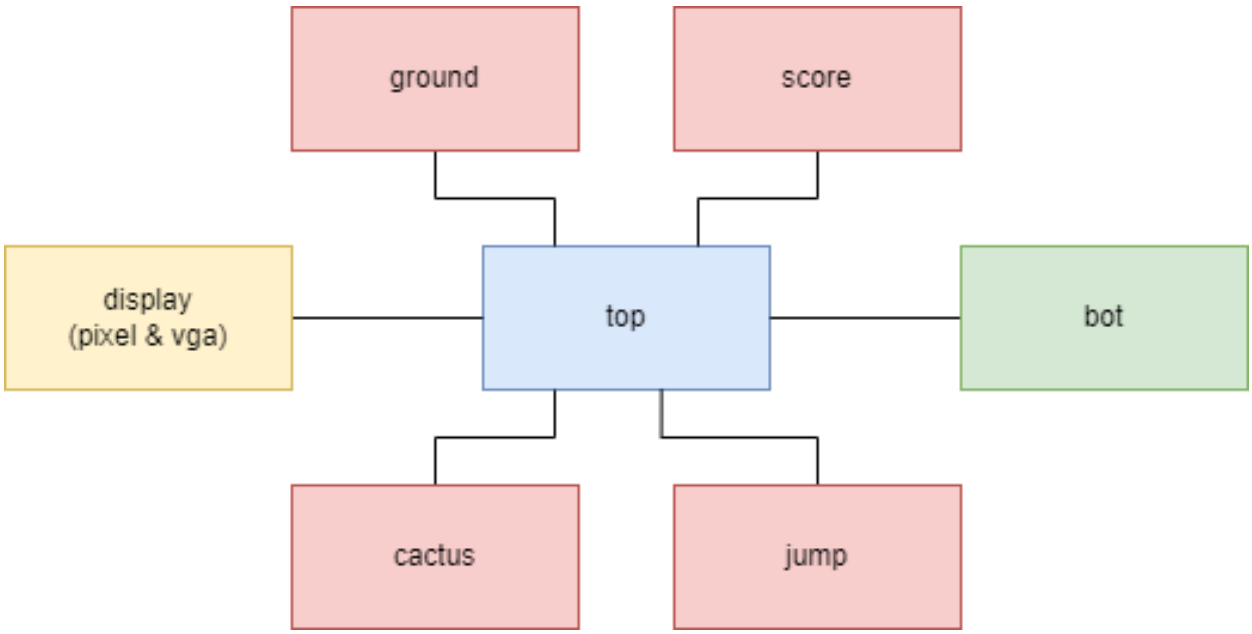
Motivation

在跳槽 Edge 前，Chrome 是最常使用的瀏覽器。既然我們熱愛 Chrome，順便喜歡他們的離線小遊戲也是相當合理的，故選擇這個陪伴小時候斷網時的樂趣作為這次製作的小遊戲。之前助教展現給我們看前幾屆做的 flappy bird Q learning 也是讓我們想做 Q learning 的原因。訊號和表現不會太複雜的遊戲也可以減低我們製作 Q learn 的難度，小恐龍遊戲便是相當不錯的選擇。

System specification

主要 module 連接

這是各 module 的連接圖，其中 top 把各 module 連結起來，並進行 state 的判定，和負責產生要求 bot 更新 Q table 的訊號。Ground、cactus、jump、score 這四個 module 會藉由 top 傳入的 state 和 vga controller 的 h_cnt、v_cnt（也就是螢幕上的座標）來產生對應的像素顏色。Bot 則是利用輸入的 state 和 Q state（更新 Q table 的訊號）來判定現在這個狀態應該跳或不跳並回傳給 top。

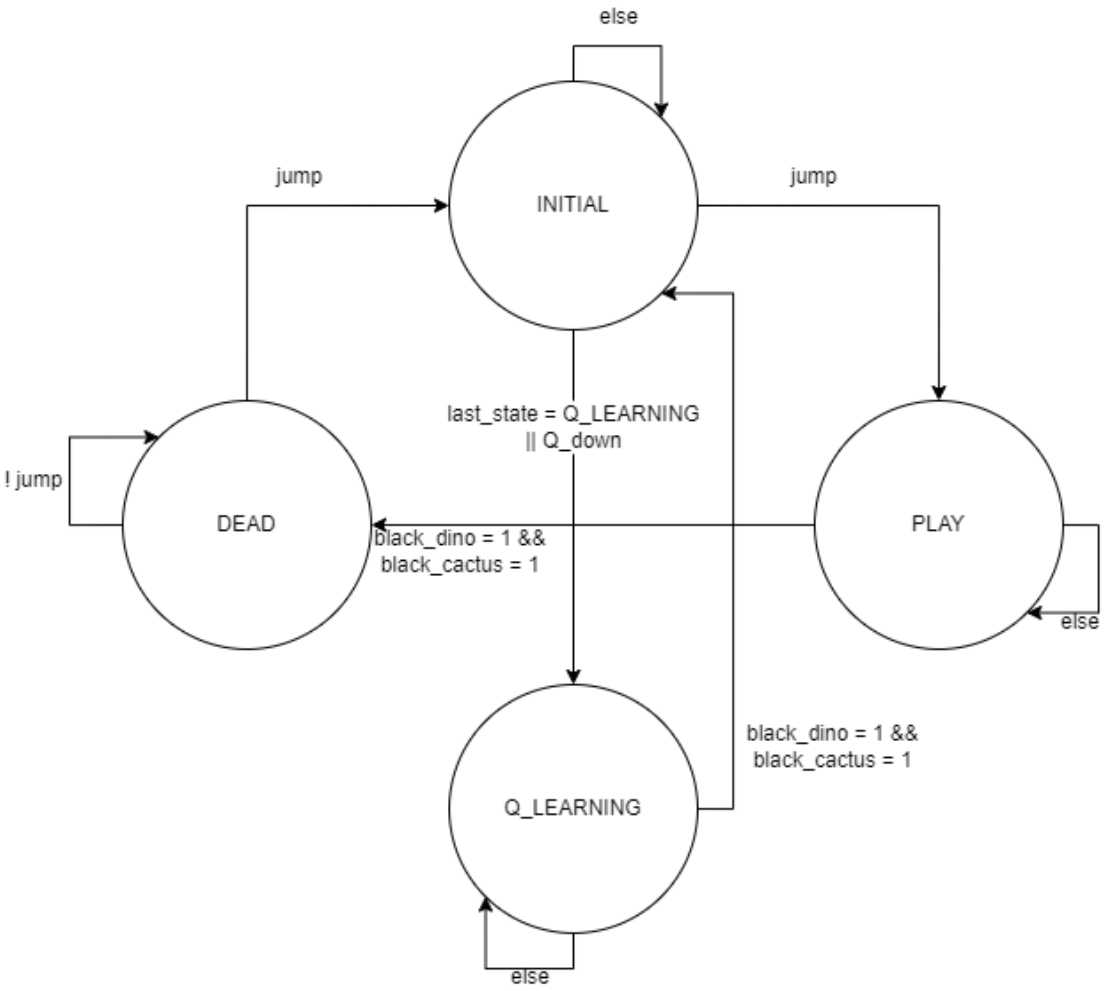


Top module

1. 處理基本輸入：debounce module, onepulse module, keyboard decoder 處理 FPGA 板中間按鈕、鍵盤 Q 鍵、空白鍵、方向鍵上的輸入。
2. 時鐘：連接 clock_divider module，取得 25MHz 和 1Hz 的 clock，並且將 vga module (助教提供的 vga controller) 的 vsync 訊號做 onepulse，取得 new_frame 訊號作為 fps clock。
3. 連接遊戲與輸出：將 ground, score, cactus, jump module 連接 pixel module，為的是取用四個遊戲 module 的輸出 black_[name] 作為產生 pixel 的資訊。

4. 連接遊戲與 bot：將 cactus, jump module 與進行 Q learn 的 bot module 連接，為的是將仙人掌的種類，位置傳給 bot module 用來更新 Q table ，做出判斷後，將 prediction 訊號傳給 jump module 來表現 Qlearn 時的跳動行為。

5. 遊戲 state transition：根據輸入訊號、遊戲 module、bot module 的輸出訊號來決定 state transition。由於玩家版本和 Qlearn 版本的 state transition 會不一樣，故使用 last_state 來記錄上一個 state，表現如下圖。



6.

7. 產生 Q state：根據遊戲 module 輸出訊號決定 Q state ， bot module 會由此決定 Q table 如何更新。共有四種狀況，00 沒有發生事情、不更新，01 是成功跳過，10 是在非跳躍狀態撞到仙人掌，11 是在跳躍狀態撞到仙人掌。

Ground module

使用 reg pattern 陣列儲存地板的外觀，使用 ground_position 紀錄地板的位置達到移動的效果，輸入 h_cnt, v_cnt (vga 的 screen 座標) ，若該位置需要顯示黑色地板，則將輸出 black_ground 設為 1。

Cactus module

連接 random module，決定隨機產生的仙人掌類型和出現時間。與 ground 相同，使用 reg 陣列儲存仙人掌的外觀，position 決定位置，輸出 black_cactus 決定是否該位置是否要顯示仙人掌。另外，cactus module 需另外將 position 和 type 輸出，用於傳送給 bot module 和決定 Q state 的資料。

Jump module

同樣使用 reg 陣列儲存恐龍的外觀，我們將恐龍的外觀分成身體(維持不變) 腳跑動(共兩種交替) 和頭(死掉時頭部一樣)。jumping 表示恐龍是否為跳的狀態，jump_time 用於計算起跳後過了多久，再使用 jump_time 計算恐龍應該出現的位置。可接收玩家的輸入 jump 或是 bot module 的指令 Q jump，來決定跳起。

Score module

用於顯示計分。一樣使用 reg 陣列先行儲存 'score' 字樣和四位數字。將 16 位二進位的 reg 分數輸出成四位十進位的分數顯示，每四個 bits 決定一個十進位的 bit，並將其做正確的進位。

Pixel_gen module

連接 vga module (助教提供的 vga controller) ，決定該 pixel 的輸出訊號，只要該點需要輸出恐龍、地板、仙人掌或分數，都將其設為黑色。

Random module

使用 32 bits 的 LFSR 來產生 8-bit random number。

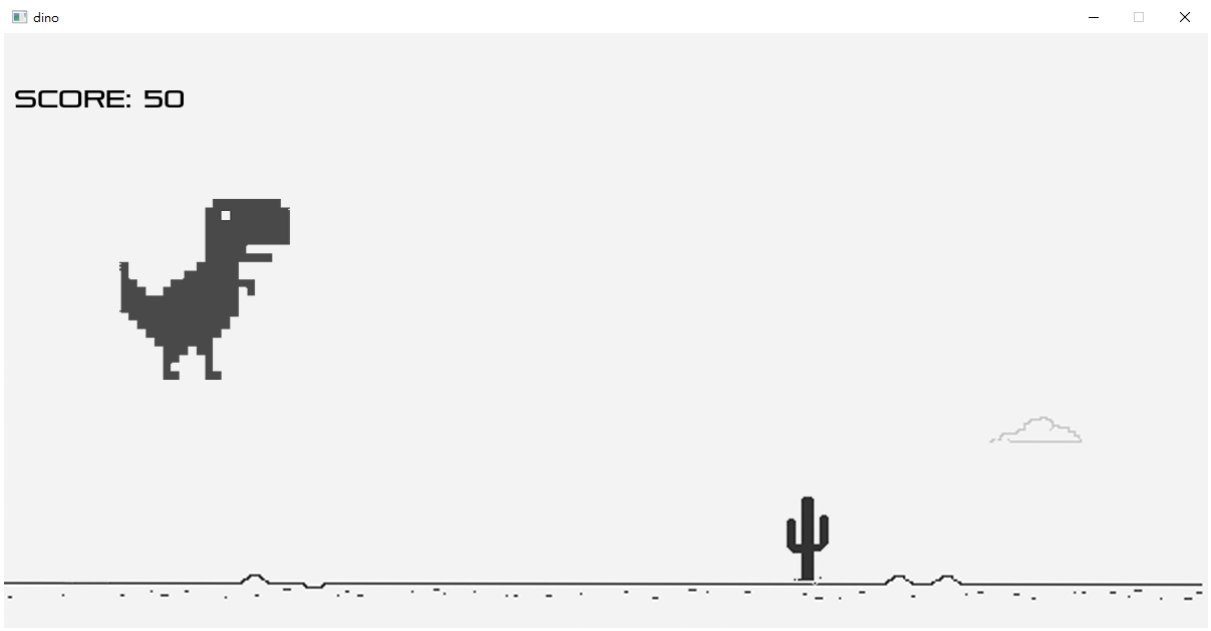
Bot module

使小恐龍隨機跳，或是輸入 top module 產生的 Qstate 來決定 Qtable 的更新方式，根據 Qtable 的結果比對決定此時小恐龍要不要跳 (set prediction to 1 控制 jump module)，Q learn 詳細運作方式將在下方相應處說明。使用 epsilon 計算 Qlearn 了多久，到達一定時間後則不進行隨機跳。

Keyboard_decoder, debounce, onepulse, vga, clock_divider module

使用助教提供的 template module 達成所需的功能。

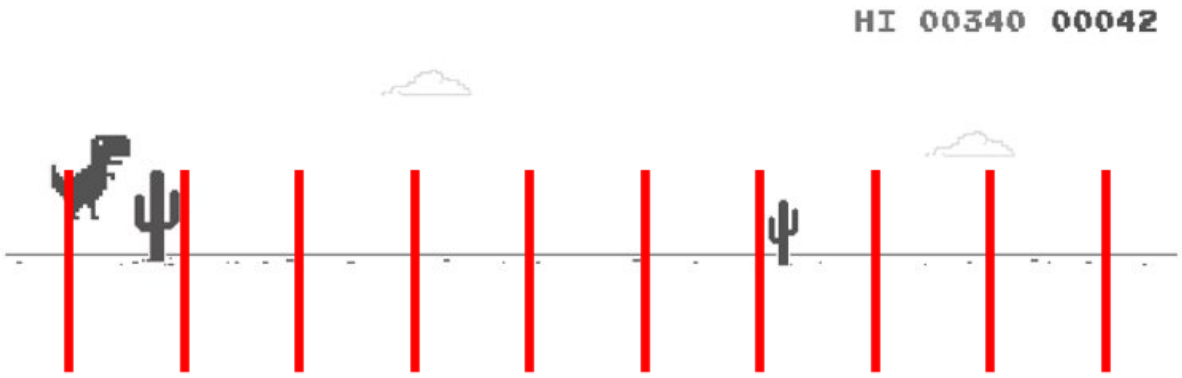
Prototypes



在寫 verilog 之前，我們先用 CPP 進行模擬，驗證我們 Q-learning 的演算法是做得起來的。主程式的部分我們直接修改我們在程式設計一做過的遊戲。至於機器人的部分我們要先簡單介紹一下 Q-learning。

Q-learning:

我們會儲存一個 Q table，也就是一個陣列，用來記錄在特定狀態（state）下做出某個動作（action）的獎勵值。在此 project 中，有兩個狀態輸入：和前面仙人掌的距離以及前面仙人掌的種類（每個仙人掌高度寬度皆不同），和兩種動作：跳和不跳。若把每個單位距離都算成一種值狀態的話，Q table 會太大，浪費記憶體資源也不好訓練模型，因此我們把畫面切成 32 等分，每段是 640/32=20 單位長。




```
int q_table[NUMofSTATE][NUMofCACTUS][2]; // 0: don't jump, 1: jump
```

當機器人執行一個動作造成狀態變化時，我們必須更新 Q table。有兩種情況會造成狀態變化，分別是成功跳過一個仙人掌（恐龍的最左邊超過仙人掌的最右邊）時和跳到仙人掌上死掉（死掉時是跳的狀態）時。第一種情況我們將對應的 Q table 的 entry 進行以下變化：Q table [距離 / Sector Length][仙人掌種類][跳] += (max value − Q table [距離 / Sector Length][仙人掌種類][跳]) << 2（也就是 * 0.25）；第二種則是：Q table [距離 / Sector Length][仙人掌種類][跳] −= (Q table [距離 / Sector Length][仙人掌種類][跳] − min value) << 2。會將變化乘上 0.25（a.k.a. learning rate）是為了讓它不要學那麼快，並藉由多次微調來達到最佳數值。另外，我們除了這兩種變化，其實還有想過第三種，也就是沒有跳就直接撞上仙人掌，但是因為此種狀況不好判斷是在哪一段距離沒有跳所造成的結果，若我們只更新離仙人掌最近的那個距離的 state，會像在直接和機器人說它應該在哪裡跳才不會撞上仙人掌，有點作弊的感覺，因此我們最後還是拿掉了。

```
39  void Bot::update(int reward, int state, int cactus, int action) {
40      game_log("UUUUUUUUUUUUUUUUUUUUUpdated!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
41      state = state / SECTOR;
42      if (reward > 0) {
43          q_table[state][cactus][action] += learning_rate * (max - q_table[state][cactus][action]);
44      }
45      else {
46          q_table[state][cactus][action] += learning_rate * (q_table[state][cactus][action] - min);
47      }
48      if (epsilon > 0 && al_get_time() - last_update_epsilon > 20) {
49          last_update_epsilon = al_get_time();
50          epsilon--;
51      }
52      dis = NUMofSTATE-1;
53  }
```

Bot::update()

為了讓機器人學習，我們不能完全讓它照著 Q table 行動，需要偶爾隨機一下。這個概念就是 Exploration v.s. Exploitation，探險和利用。而我們使用一個參數ϵ來調整這兩者的比例。一開始$\epsilon = 10$，也就是有 10/256 的機率會直接跳起來。但因為我們不想讓學成後的機器人隨機亂跳，我們每經過 20 秒會將ϵ減 1，直到它歸零為止（48～51 行），也就是在開始學習 200 秒後，機器人會照著它的 Q table 採取最佳策略。

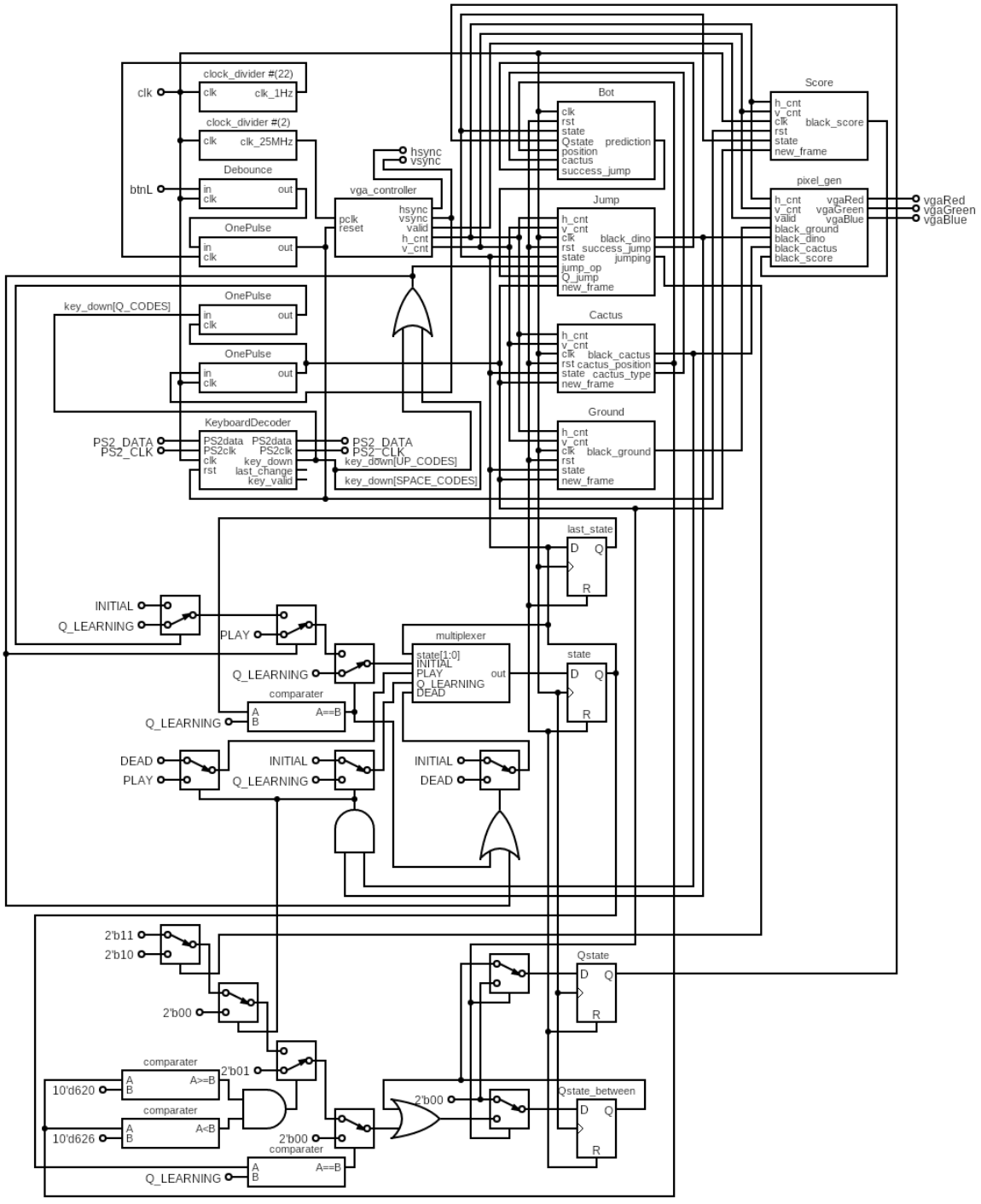
```
29  bool Bot::predict() {
30      game_log("q_table[%d][%d][1] = %d, q_table[%d][%d][0] = %d", dis / SECTOR, front_cactus, 
31      if (uni(generator) < epsilon) {
32          return true;
33      }
34      else {
35          return q_table[dis/SECTOR][front_cactus][1] > q_table[dis/SECTOR][front_cactus][0];
36      }
37  }
```

Bot::predict()

以上就是我們用 CPP 模擬的結果，和 verilog 不太一樣的地方是：在 CPP 中我們透過呼叫函式來更新 Q table，而在 verilog 實作中，我們有一個訊號叫 Q state，Q state 會在需要更新時拉起，對應到更新情況的數值，因此在 bot 的 module 中我們只需要檢查 Q state 是否為零，若不是的話我們要根據其數值更新 Q table。

Implementation

Top

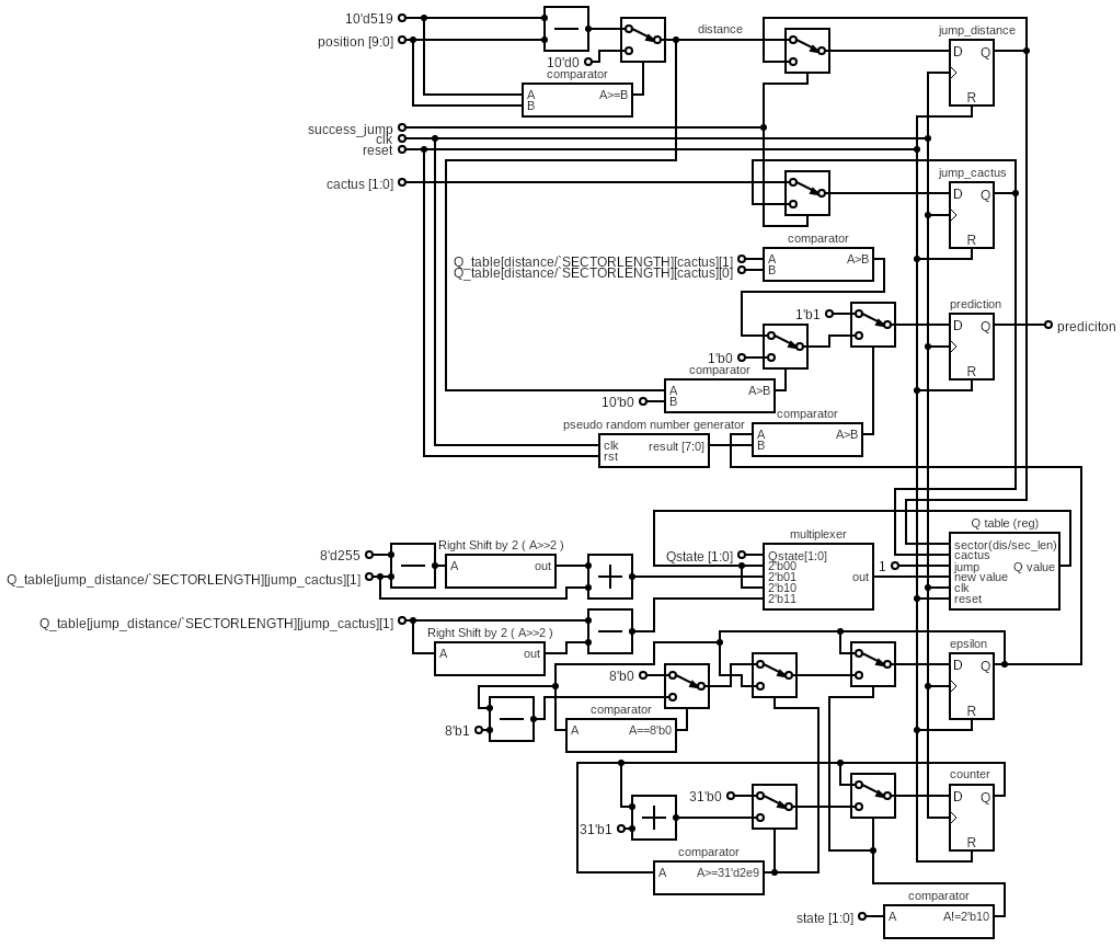


Top module 主要負責三個功能：串起各 module、更新遊戲 state、更新 Q state。比較需要注意的地方是：由於各遊戲 module（Jump, Cactus, Ground, Score）基本上都是在 VGA controller 輸出的 vsync 拉起時才會更新，因此我們將 vsync 經過 one pulse 處理輸出 new_frame 訊號（可以看做是 FPS）傳入遊戲 module。而 h_cnt 和 v_cnt 是現在螢幕上顯示的座標，各遊戲 module 會根據這兩個 signal 輸出現在這個像素應該是黑的還是白的。

遊戲 state 的部分就照著 specification 的 state diagram 接。我們用一個 DFF 來紀錄上一個 state 是什麼（last_state），因為我們設定在 Q learning 時死掉的話會直接重來（進入 INITIAL state），不用等待玩家輸入，也就是在 INITIAL state 時如果 last_state 是 Q_LEARNING 的話就會直接回到 Q_LEARNING state。

Q state 的部分比較特殊一點，舉例來說：在 frame 和 frame 之間，若小恐龍和仙人掌撞在一起，可能會有好幾個像素重疊，在掃過這些像素時皆會判定成死亡，Q state 會拉起來好幾個 clock cycle，由於我們希望這個更新訊號只要拉起一個 clock cycle（否則 Q table 會更新好幾次），由於不更新的 Q state 是 2'b00，我們用 Qstate_between 和 bitwise or 來記錄 frame 和 frame 之間的所有狀態（Qstate_between <= Qstate_between | next_Qstate），並在新的 frame 時再輸出到 Q state 一個 clock cycle。

Bot



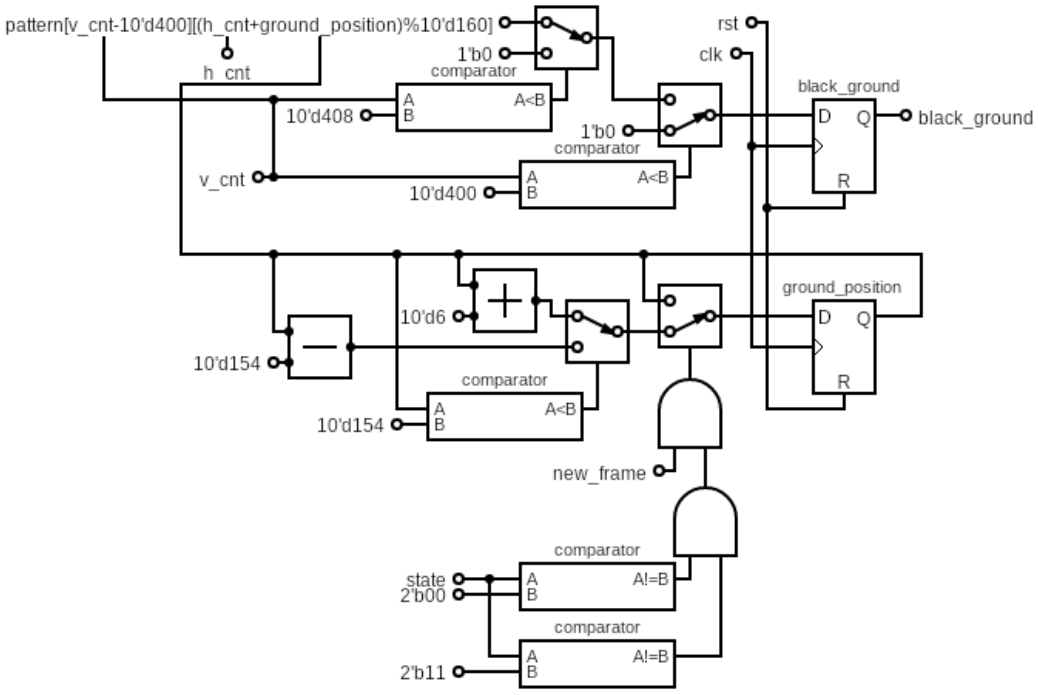
當小恐龍成功跳起來時（success_jump == 1’b1），我們將當時和前面仙人掌的距離和種類記錄起來，以供更新 Q table 的時候參考。

輸出的 prediction 則是會根據 epsilon 和隨機產生的數字，若 rand > epsilon 則輸出 1’b1，否則根據 Q table 和現在的位置及仙人掌輸出。

當輸入的 Q state 拉起成 2’b01（跳過仙人掌）或 2’b11（跳到仙人掌上死掉）時，我們會更新對應的 Q table entry（Q_table[jump_distance/`SECTORLENGTH][jump_cactus][1]）。

最後，我們用一個 counter 來記錄開始 Q-learning 後經過的時間，每過 20 秒，我們將 epsilon 減 1，直到 epsilon 歸零為止。

Ground

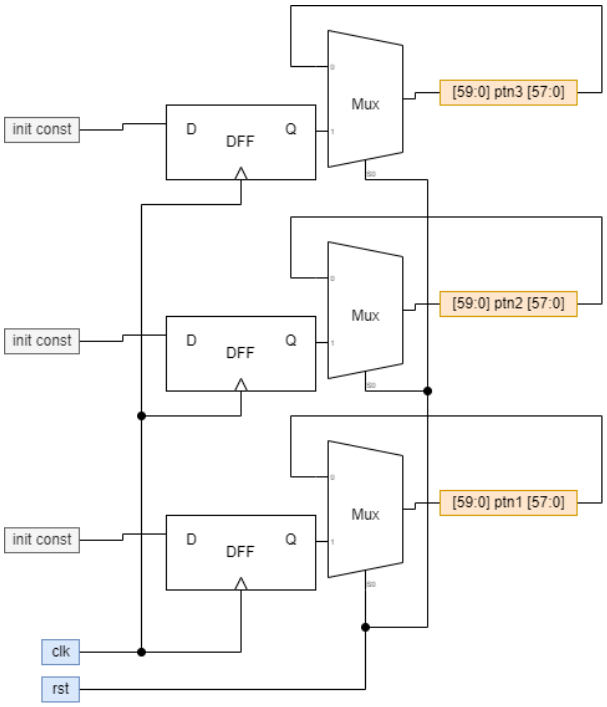


在每次更新畫面時（new_frame == 1’b1）且 state != INITIAL（2'b00）和 Q_LEARNING（2'b11）時，我們將 ground_position 加 6，也就是移動地板 6 個像素。

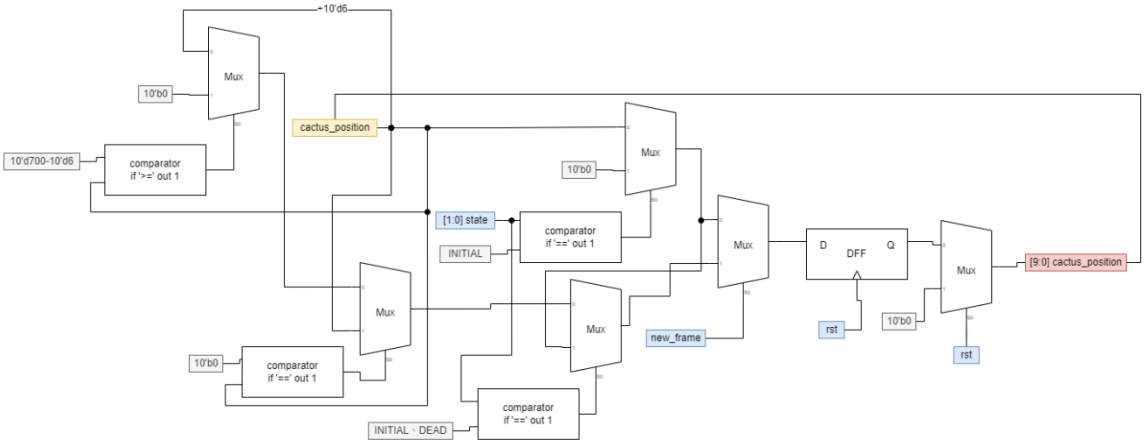
要輸出的 ground_black，我們則會先檢查現在垂直座標（v_cnt）是否介於 401 和 408 之間，若否則輸出 1'b0，若是則參考 pattern（先寫好的圖片）輸出

pattern[v_cnt-10'd400][(h_cnt+ground_position)%10'd160]。

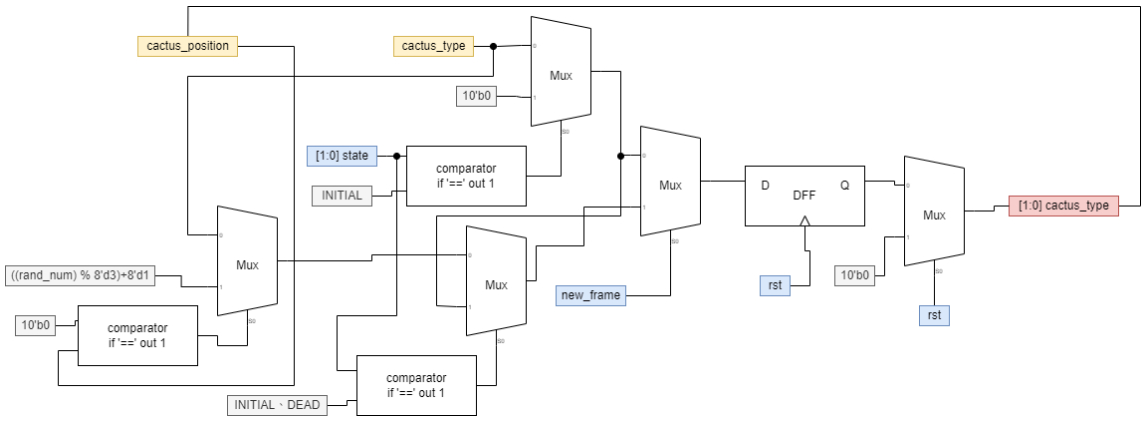
Cactus



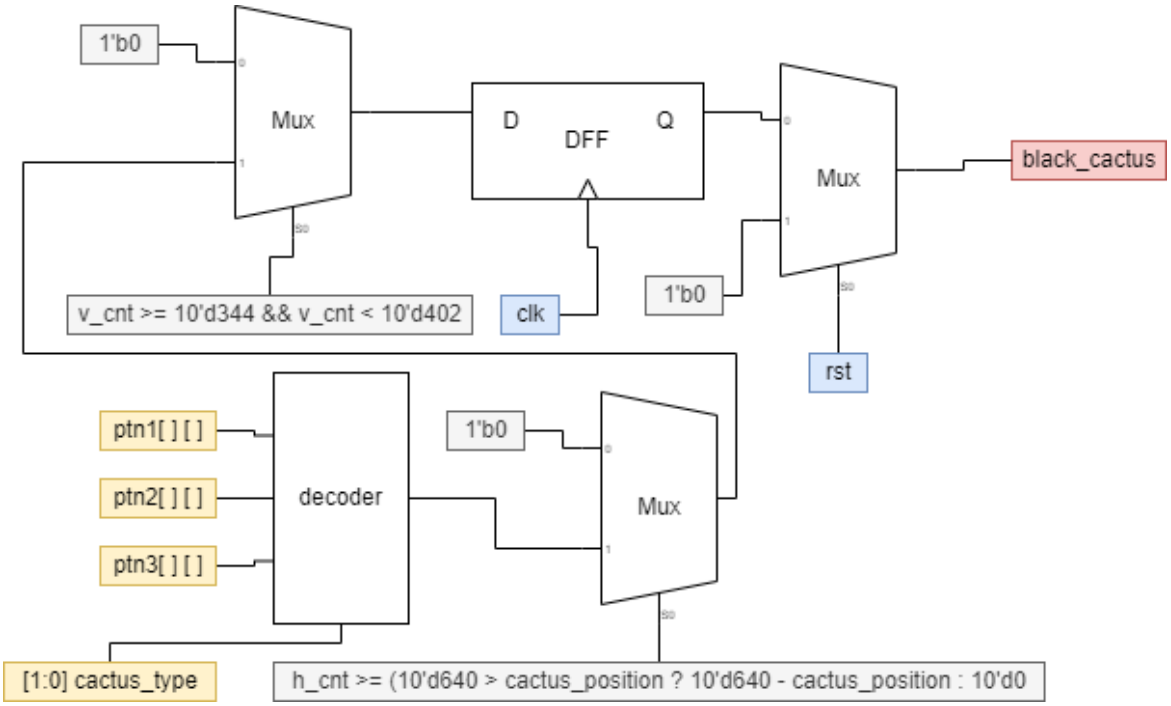
本來使用 intial block 初始三個 ptn ，但因為 synthesis ，故使用 sequential 來初始三個 ptn array。ptn array 用 01 來儲存仙人掌的外觀，1 表示為需要印出黑色的部分。



上圖是 output cactus_position 的部分，可以看到在遊戲 clock triggered 時，判斷遊戲 state 是否為 inial 或 dead，如果是的話，則需要計算 position。position 為仙人掌最左邊的座標，從右側螢幕為 0 開始，若 position 下一次的移動大於 700(640 螢幕寬+60 仙人掌寬)，表示仙人掌最右側消失在螢幕，則可以將其位置歸零。根據 wait_time(由 random 決定，在上一個仙人掌消失後隨機 0~2 秒)決定何時出現開始出現且移動。

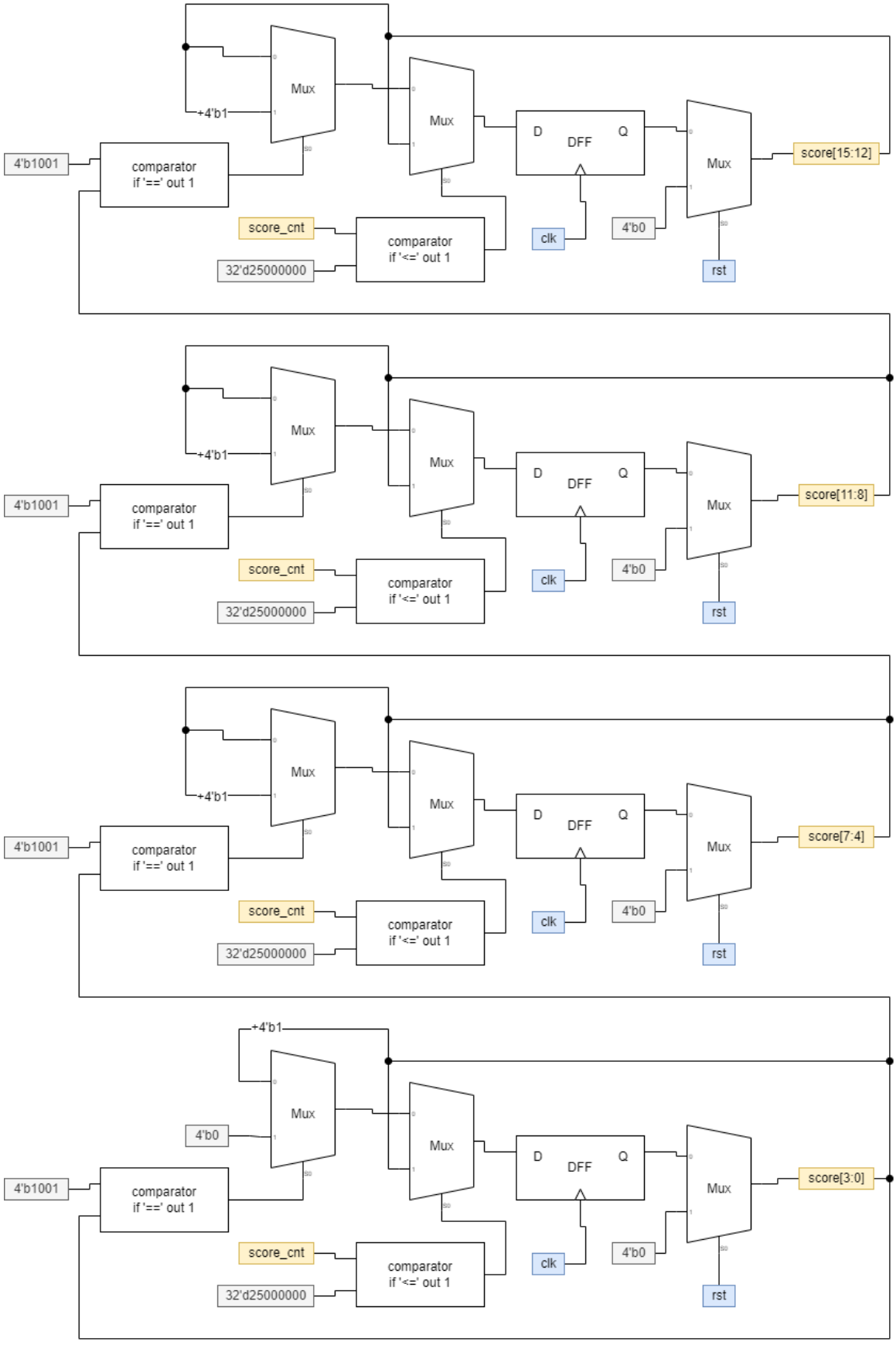


上圖是 output cactus_type 的部分，每次 position 歸零時，我們需要隨機產生一個 cactus_type 決定產生哪一種仙人掌。隨機的 rand_num 我們透過連接 random module 來取得。



上圖為 black_cactus 輸出，藉由 h_cnt, v_cnt, cactus_position 可以決定 vga 現在所跑的這格是否要輸出黑色的仙人掌部分(黑色則為 1)。我們使用兩個 MUX 可以加快判定，仙人掌只會出現在固定的 y 座標範圍內(仙人掌不會跳)，所以可以先檢查是否在合理的 y 座標範圍內。

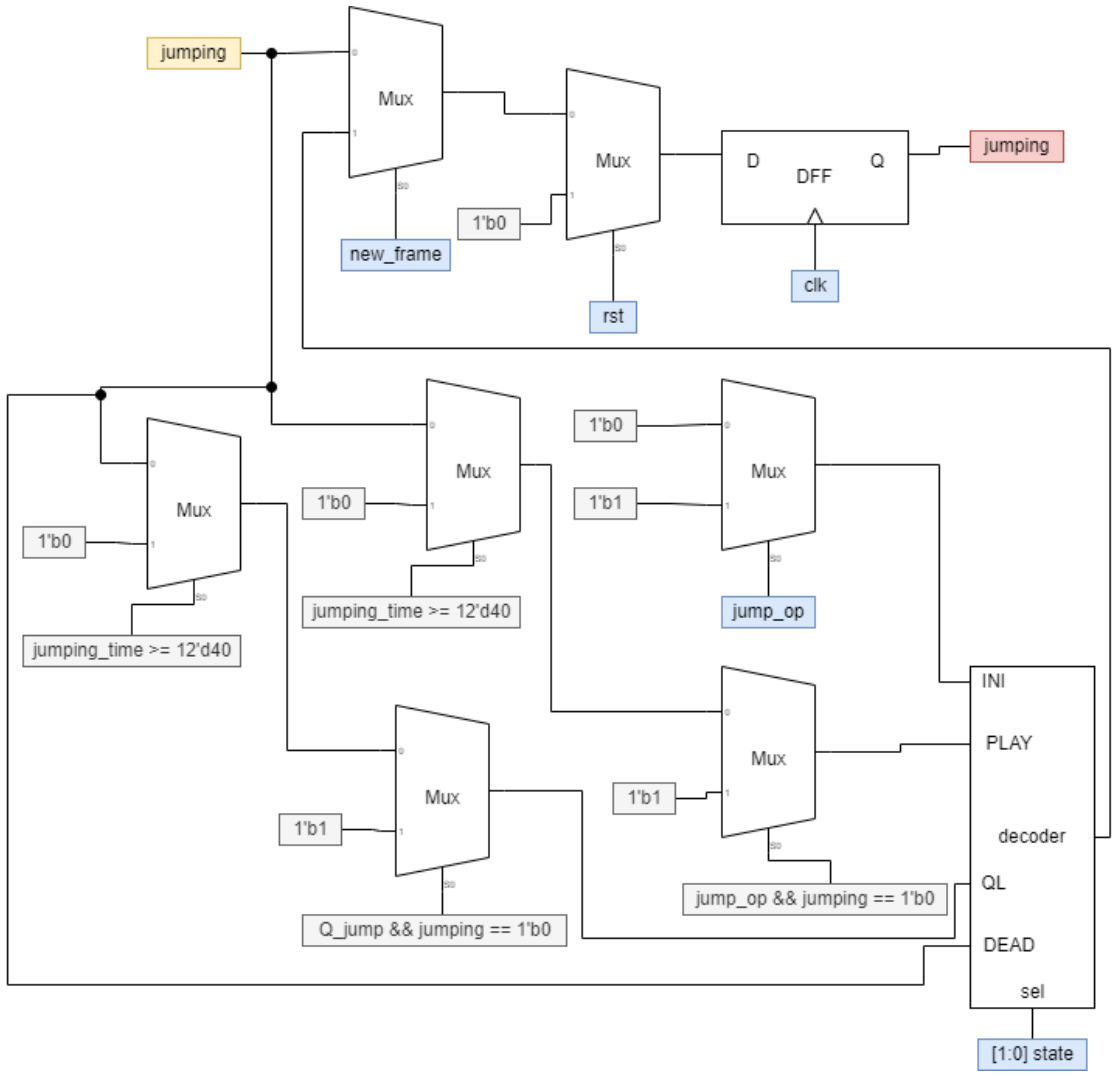
Score


Mux
+4'b1
Mux
D
DFF
Q
Mux
score[15:12]
4'b1001
comparator
if '==' out 1
score_cnt
comparator
if '<=' out 1
32'd25000000
clk
4'b0
rst
Mux
+4'b1
Mux
D
DFF
Q
Mux
score[11:8]
4'b1001
comparator
if '==' out 1
score_cnt
comparator
if '<=' out 1
32'd25000000
clk
4'b0
rst
Mux
+4'b1
Mux
D
DFF
Q
Mux
score[7:4]
4'b1001
comparator
if '==' out 1
score_cnt
comparator
if '<=' out 1
32'd25000000
clk
4'b0
rst
+4'b1
Mux
4'b0
Mux
D
DFF
Q
Mux
score[3:0]
4'b1001
comparator
if '==' out 1
score_cnt
comparator
if '<=' out 1
32'd25000000
clk
4'b0
rst

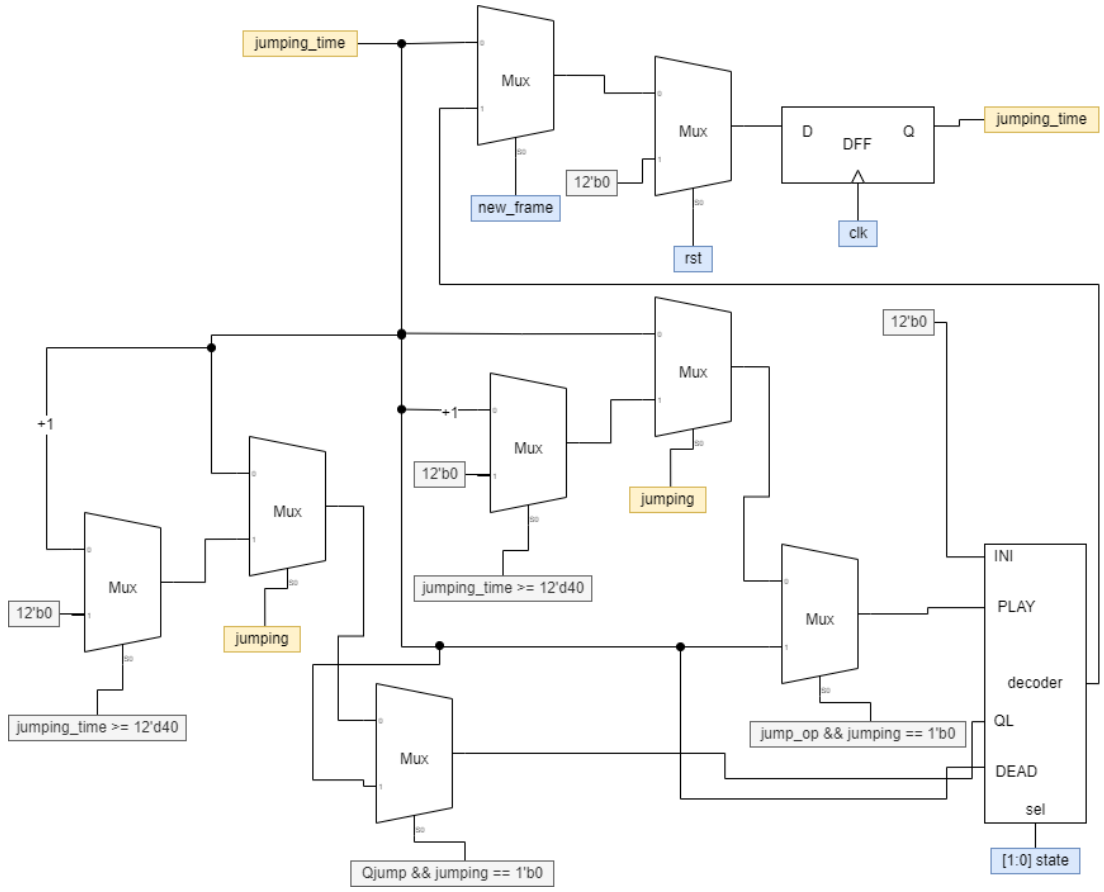
上圖是將 16-bit 紀錄分數的 reg score 四個 bit 一組，將二進位轉成十進位。score_cnt 控制加分的速度，每次都是 LSB(score[3:0])+1，當 LSB 為 9 時，歸零自己且將下一位+1，如此重複往高 bit 計算。

在 score module 中，我們依舊將'score'字樣和四位數字分別初始化到相應紀錄的陣列中，在 input h_cnt, v_cnt 到達應顯示字樣或分數的座標時，將 output black_score 設為 1。

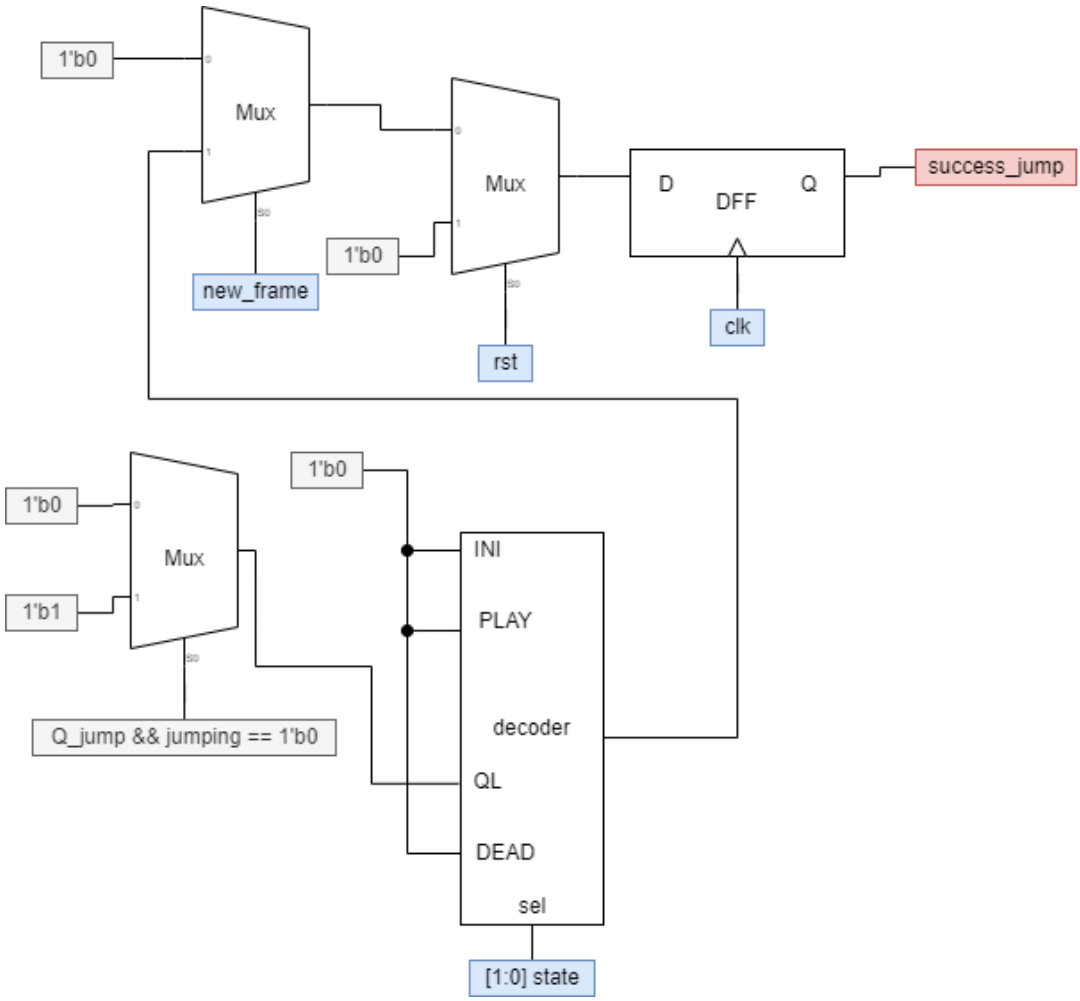
Jump



上圖為 output reg jumping，整個跳躍狀態時值為 1，否則為 0。在 INITIAL state 接收到 jump_op 時起跳 (加上 top module 的 state transition，在 INITIAL 狀態按空白鍵或方向鍵上，會開始遊戲且起跳)。而在 PLAY state 和 QL state 則是分別以接收 jump_op(方向鍵上或空白鍵)、Q_jump(由 bot module 產出)決定 jumping。而當 jump_time 到達 12'd40 時則設回 0(跳動結束落地)。



上圖為 jumping_time 的更新，表現開始 jump 之後過了多久時間，40 個 fps clock 後落地。



上圖為 output success_jump, 主要功能為將 Qlearn 時的起跳瞬間傳給 bot module。

Output black_dino 則是如同另外幾個遊戲 module，判斷當前座標在初始外觀陣列+更新當前位置後是否需要印出黑色恐龍外觀。

Random

32-bit 的 LFSR，reset 時初始為 32'b10111000110101101011011010111100，在第 1、2bit 和第 22、32bit 處進行 xor，並取第 19、17、13、11、7、5、3、2bit 作為 8-bit output result。

Pixel_gen

由於螢幕只有出現黑色或白色，故將 {vgaRed, vgaGreen, vgaBlue} 在 black_[module_name] 任一等於一時，設為 12'hfff (黑色) 而其他狀況則設為 12'b0。將 pixel_gen 與助教提供的 vga_controller 連接後，則可以產生正確的 vga 顯示器訊號。

Experimental Results

經過一連串的測試和沒日沒夜的 debug 之後，小恐龍終於照著我們預期的跑了。在遊玩中，小恐龍能正常地跳和正常地死掉，Q-learning 的效果也比預期的好，經過 200 秒後它基本上就不會死掉了，我們甚至還看過它玩到分數 overflow（>9999）。

Problem Encountered

軟體和硬體在變數處理上非常不一樣，除了需要注意 10 進位和 2 進位，還要處理小於 0 時會變成補數的問題(如果這個時候用 unsign 比較，就會變成永遠不小於 0)。fps clock 和 fpga clock 的問題，有些 signal 只會拉起一個 fpga clock cycle，而有些東西在 fps clock 拉起時才會更新，造成兩者對不上的問題。剛開始使用的 8-bit LFSR 在經過一段時間後，變成不太 random。Qlearn state 的轉換問題，也讓我們處理了一段時間，因為在 Qlearn 死掉時只花一個 fpga clock cycle 進入 INITIAL 後繼續回到 Q state 訓練，就會有上面提到和 fps clock 對不上的問題。

Future Works

擴充遊戲表現，如增加下蹲指令、背景增加鳥、雲、可以從白天黑夜轉換、增加音訊、速度會段落式加快。

根據遊戲的擴充功能改寫 Qlearn，可以學習下蹲，可以根據速度更新判斷等等。

Conclusion

在這個 project，我們學到了在硬體寫遊戲的基本觀念和在硬體上套用 Q learning 的觀念，了解到硬體與軟體真的相差非常多，很多訊號傳遞不是像軟體一樣一個參數就飛過去了，還要考慮各種 clock 影響，都是在寫軟體時不會遇到的問題。這個學期修了這門課學習到了很多新的知識，更佳了解一些軟體表現在硬體上的實作和不同的思考方式。感謝教授一學期辛苦認真的教學。尤其感謝助教，我們有時候自己看自己的圖或 code 都會看到眼花撩亂了，何況助教要看好幾十份不同的圖和 code，辛苦了。

Contribution

遊戲：阮柏諭；Q learning：侯皓予

Report：

阮柏諭：Motivation, System Specification, Implementation (cactus, score, jump, random, pixel_gen), Problems Encountered, Future Works, Conclusion

侯皓予：Introduction, Prototypes, Implementation (Top, Bot, ground), Experimental Results, Contribution