

**111-1**

**1111 網路程式設計快艇骰子**

**Class:資訊三丁**

學號	姓名	班級
D0957318	鄧琪融	資訊三丁
D0972425	許祐瑋	資訊三丁

## 摘要：

快艇骰子是《世界遊戲大全 51》中收錄的遊戲，是使用骰子進行的撲克風遊戲，只需要有 5 顆骰子和記分板就可以達到 2~10 人的同樂小遊戲，遊戲中非常注重運氣，我們希望可以加入籌碼來達到刺激又好玩的賭博連機小遊戲。而我們的目的是建立一個可連網的線上遊戲，分為 Local 以及 Multiple mode，可以讓使用者依據當前的條件選擇遊玩方式。

## 人力分配：

整個遊戲目前主要分成 2 大部分，一個為前端，演示骰子的投擲過程，以及玩家所使用的 UI 介面，掌握大多數的玩家遊戲流程，這部分由鄧琪融來負責，另一個為後端，負責產生遊戲內部真正的結果，例如骰子所應顯示的數字、或遊戲進程控制，這部分由許祐瑋來負責，最後進行整合來將程式前後端進行整合以及通訊的部分，就一樣由鄧琪融來進行。

## 文獻：

主要參考的是世界遊戲大全 51 中的一款名為“快艇骰子”的遊戲，遊戲中的骰子與我們最大的差異就是計分的規則了。在世界遊戲大全中，因為回合數縮減到了 12 回合，因此下排記分板的三骰同花(Three of kind)也就被取消了。



另外我們也參考了 cardgames.io 上的快艇骰子，在這個網站中的規則是完全與最初始遊戲的規則一樣，我們也採用了相同的規則。但是在這個遊戲中只支援了一名玩家與一名電腦。我們在這個版本的基礎上增加了玩家，我們的程式允許最多四名玩家加入遊戲



## 研究動機：

我從小就在任天堂的遊戲中度過，各種大大的小的遊戲我都會一玩再玩，而在最近的 switch 中出現了這樣的一款遊戲，稱作快艇骰子，其遊戲內容相當的簡單，不需要任何操作，全憑運氣以及經驗的判斷，並且規則簡單明瞭，與他人同樂時幾乎沒有學習期，可以非常快就上手，且深深投入其中，可問題在於 switch 當初在設計這款遊戲時，並沒有加入類似獎勵的機制，缺乏使玩家繼續遊玩的動力，也就是因為這樣，我希望可以實踐一款擁有獎勵機制的遊戲，吸引他人一場一場的遊玩，可以一起享受在快艇骰子的樂趣中。

## 程式功能：







實踐骰子的投擲情況，並且給予兩次重骰的機會，並且同時進行計算得分以及獲取最終骰子的得分情況，至於人數方面我們設定為 4 人連線，若人數不足時，可以選擇使用電腦作為代打，所以可以達到 1~4 人的連機遊戲，對於玩家我們會有獨立的資料庫系統，保存玩家資料，包括:姓名、金幣.....等。

### 規則:

整個遊戲總共由 13 個回合所組成，在遊戲中玩家必須以組合出最大的數字為目的，並且透過骰子的組合取得相對應的積分，在遊戲的最後取的積分獲得最終勝利。在每一回合中玩家可以擁有三次投擲骰子的機


會。除了第一次的投擲之外，剩下的兩次投擲都可以選擇只投擲部分的  
骰子來獲得更高的積分，但是需要注意的是，只要第三次重擲骰子之後  
就必須選擇一個格子將積分登入到計分表中。

上排：

名稱	說明	分數	範例
Aces	點數為 1 的總和		 獲得 2 點
Twos	點數為 2 的總和		 獲得 2 點
Threes	點數為 3 的總和		 獲得 9 點
Fours	點數為 4 的總和		 獲得 8 點
Fives	點數為 5 的總和		 獲得 20 點
Sixes	點數為 6 總和		 獲得 30 點

● 下排:

名稱	說明	分數	範例
Three of kind	要有三顆骰子相同點數	所有骰子點數相加	 獲得 14 點
Four of kind	要有四顆骰子相同點數	所有骰子點數相加	 獲得 9 點
Full House	三個相同點數的骰子加上另外兩個相同點數的骰子	2 5	 獲得 25 點
Small Straight	四顆骰子組成連續數字	3 0	 獲得 30 點
Large Straight	五顆骰子組成連續數字	4 0	 獲得 40 點
Yahtzee	所有的骰子有相同的點數	6 0	 獲得 60 點

Chance	所有的骰子	所有骰子點數相加	 獲得 16 點
--------	-------	----------	--

特殊規則：

Yahtzee 被填了兩次以上，並在 Yahtzee 的方塊並未填 0 的狀況

下，那麼在 Yahtzee 的方塊裡會加 100 分，當上排分數超過 6 3 分

時，即可在

Bonus 中填入 3 5 分。

## 工作時程：

起始日期	結束日期	實踐目的
9/26	9/30	整體架構構思
10/3	10/10	前端 UI 設計
10/10	10/17	前端程式碼化&後端構思
10/24	10/28	前端完成&後端程式碼化
11/1	11/11	結合前後端
11/12	11/20	前後端除錯及開發登入介面

11/20	11/27	實踐遠端連線及應用登入介面
11/27	12/1	實作等待大廳及套入遠端連線介面
12/2	12/9	全面除錯及壓力測試

## 程式架構大綱：

這隻程式將分成數個 class 來達成我們主要的遊戲內容呈現，其中包含：

**Game:**負責運行程式後端，進行運算得分及產生內部結果，再透過 global

參數 out 匯出至 **class game**，是整個程式的靈魂所在 **game:**負責運行程式

的前端，持續刷新畫面，給予使用者提示，並操作 **class Dice** 來進行骰子的

動作，包含了骰出點數的呈現，以及提供使用者見面供使用者可以選擇重骰

或保留想要的點數。

**Dice:**負責管理使用者的各項資訊，實作個別 GUI 介面以及連接，使使用者

可以達到想要的目的，作為一款骰子為主的遊戲，骰子具備動畫且呈現效

果，需同時與 **class game** 和 **Game** 進行互動，確保兩者資訊一致，才能使

遊戲內部與外部 UI 呈現正確，避免使用者誤判。



## 程式架構內容：

Client(Dice):

Class login\_lobby:

實作登入大廳，提供介面供使用者進行登入驗證，使用者可以進行註冊或登入，點擊方框時可以變更方框顏色，告知使用者目前正在使用哪個輸入框，以免用戶重複註冊，但若使用者重複註冊時，會回傳警告框，並清空輸入框資料。

Class waiting\_lobby:

實作等待大廳，提供使用者資訊，顯示金錢以及名稱，並提供使用者可以選擇 Local mode 或 Multiple mode，當點擊 Local 時，可以選擇遊玩人數，然後即可使用開啟 class main\_dice 來開始遊戲，若是選擇 Multiple mode 則會使用函式 connect 透過 request 去訪問 Server 來進行配對，其中 connect 時會去檢測該玩家當前的遊玩狀態，若為正在遊戲中，或者配對中就會取消該用戶端的配對，避免該玩家同時加入遊戲，導致惡性刷錢。若使用者的金錢數低於 300 美元，則忠於原作，不給參與遊戲，再次點擊畫面就會被強制退出。

Class Dice:

實作遊戲中最重要的物件骰子本體，事先讀入骰子靜態的圖片，以及骰子轉動時所需的動態圖片，並將初始值設為 1，透過內部函數撰寫 move 可以使骰子的圖片變更成下一張骰子的動態圖片，這個函式會用在函式 roll 中，將骰子的 rolls 參數設定為 True 及開始進行旋轉，且透過隨機產生的次數來使每個骰子

的骰動次數不同，同時每次都觸發 move，再透過撥放音效，就可以達到骰子從靜態變成動態轉動，增加畫面的多變性。

## Class Game:

整個程式的主 UI 介面，並且掌管 Client 的流程，在該玩家的回合時進行骰子骰動次數的計算，並且確定玩家點擊時所造成的回饋(如轉動骰子獲得新的點數，將骰子上鎖以避免想要的骰子重骰，並加上上鎖的圖示告知用戶該骰子已被上鎖)，在回合的最後確認玩家選擇的計分方式，若當前非玩家的回合，則鎖定畫面，將畫面的所有點擊框都失效，避免造成玩家搶奪回合的發生，右方則會顯示該玩家的計分方式選擇，並將計分寫在上方，若點在已有的分數格上則會視做無效操作。該輪遊戲結束後，會將結算分數顯示，告知名次，若該用戶為第一則增加 600 元，若該用戶為第二則增加 300 元，若該用戶為第三則減少 300 元

若該用戶為第四則減少 600 元。若在遊戲進程中，玩家離開當局，則會傳

送訊號告知 server，將該玩家改成電腦接替繼續進行遊戲，以免形成僵局。

Class Multiple\_temp:用來暫時儲存多人連線時送來的指令，該 class 會在

啟用多人模式時開啟一個新的 thread 自動定時 0.5 秒向伺服器索取資料。

## Main:

整個遊戲的進程都將會維持著下方的迴圈運作。

```
while(True):
    if user == {}:
        login_lobby()
    grade = [[0 for _ in range(14)] for _ in range(true_player)]
    waiting_lobby()
```

```
print(cpu_player)
if Local_sign:
    back_game=main.main_dice(str(true_player),cpu_player)
if Multiple_sign:
    back_game=Multiple_temp()
back_game.setDaemon(True)
back_game.start()
Game().run()
```

主要是先判斷是否有進行登入，沒有則導入登入介面，登入了就進入大廳，

選擇遊戲模式後就進入主遊戲介面。

### Server(Main):

class game:

遊戲處理中心，由 main\_dice 提供遊戲必要的輸入輸出服務，在交由此 class 處理過後再交由 main\_dice 輸出

roll\_dice:

使用 random 函數模擬骰子投擲，並且回傳一組 list，裡面為骰子骰出的結果

re\_roll\_dice:

讀取使用者的指令並且重新投擲相對應的骰子，回傳值為經過重新投擲骰子過後的結果

count\_score:

當使者骰出骰子，並且輸入成績之後，系統會交給此函式使用者的成績表、骰子數值以及使用者欲數入表格的部分。此函示會判斷其合法性，並且回傳相對應的成績。輸入值 fnc 對應的欲輸入成績如下表

fnc 輸入值	對應表格
1	Aces
2	Twos
3	Threes
4	Fours
5	Fives
6	Sixes
7	Three of kind
8	Four of kind

9	Full House
10	Small Straight
11	Large Straight
12	Yahtzee
13	Chance

經過運算之後此函式會回傳一組字串，此字串由一個符號接續數字或文字組成，所有可能產生的回傳值如下表

回傳值	說明
+XX	成功 XX 為數字，表示該表格填入的分數值
-signed	該欄位已經輸入過分數了，主程式會要求使用者重新輸入成績
-	經果判定之後骰子無法滿足條件，所以無法計分
--	發生了未知的錯誤，導致此函示無法繼續。通常是因為錯誤的輸入值導致。

cpu\_count\_score:

協助電腦玩家判斷當前骰子能得到的成績，用以判斷當前最適合做的動作。

cpu\_act:

在此函示中，電腦會試圖找出所有合法操作的組合，並且計算所有組合中的期望值，最終做出期望值最高的動作。

Class main\_dice:

骰子遊戲的主要部分，負責接受發送使用者指令，維持遊戲執行規則等工作

get\_q: 將數入的變數變成可以處理的資料

put\_q: 將輸出的資料變成外部可以判讀的資料

run:

為遊戲的外層大框架，會呼叫上述提及的函式庫與 class 來維持遊戲運作。

## 傳輸協定:

我們這次所使用的傳輸協定為 REST API(Representational State Transfer API)，該架構支援大規模的高效能和可靠的通訊。可以輕鬆實作和修改，為任何 API 系統提供可視性和跨平台可移植性。我們透過 REST API 來傳輸 Client 和 Server 的訊息。Client 通常會嘗試丟入 request 若伺服器不在則會顯示 server crash。

登入:Client 向 Server 進行 request，Server 在資料庫中尋找是否有符合的使用者，並回傳使用者資訊。在 Client 端為會去判斷，若有該使用者就顯示 login successful 不然就會顯示 no account

Server	Client
<pre>@API.get("/accounts") def get_account():     param = request.args.get('name')     print(param)     with open(ACCO_FILE) as fp:         ACCOUNTS = json.load(fp)     if(param == None):         return jsonify(ACCO_FILE)     else:         RET_COMP = []         for temp in ACCOUNTS:             if(temp["name"] == param):                 RET_COMP.append(temp)         return jsonify(RET_COMP)</pre>	<pre>#login try:     my_params = {}     my_params["name"] = self.text     response = requests.get(URL_ACCOUNT, params = my_params)     print(response.status_code)     print(response.headers)     json_rec = response.json()     for item in json_rec:         user=item         MessageBox(0,"Login Successfull", "Yahtzee",MB_OK)         return     MessageBox(0,"No account", "Yahtzee",MB_OK) except:     MessageBox(0,"Server is crash", "Warn",MB_OK)</pre>

註冊: Client 向 Server 進行 request，Server 將該資料庫先上鎖，再至資料庫中尋找是否已有相同名子的使用者，若有就告知 Client 該名子已被使用，Client 端就會彈出視窗告訴使用者。若該使用者名稱有效則會分配一唯一 ID 最後在資料庫進行登記，並給予基本金額 1500 美元。Client 端則顯示成功。

Server	Client
<pre>def register():     if request.is_json:         lock.acquire()         new = request.get_json()         new["id"] = find_next_id()         if find_same_name(new["name"])==True:             print("same name")             return {"error": "Account is same"}, 400             #尋找是否相同         new["dollar"]=1500         ACCOUNTS.append(new)         with open(ACCO_FILE, 'w') as wfp:             json.dump(ACCOUNTS, wfp)         lock.release()         return new, 201     else:         return {"error": "Request must be JSON"}, 415</pre>	<pre>#register new_dict = {} new_dict["name"] = self.text response = requests.post(URL_ACCOUNT, json=new_dict) print(response.status_code) print(response.headers) print(response.text) if response.text.find("erro")!=-1:     MessageBox(0,response.text[14:len(response.text)-4:1],"Warn",MB_OK) else:     MessageBox(0,"Register Successfull", "Yahtzee",MB_OK)</pre>

更動金錢:當遊戲結束時會變更金額，Client 會告知 Server 自己是誰，要變更多少金額，Server 在處理時會先將該區間 lock 住，避免資料遭到重複覆蓋的問題。

Server	Client
<pre>def put_account():     if request.is_json:         lock.acquire()         new = request.get_json()         uid=new["uid"]         money =new["money"]         with open(ACCO_FILE) as fp:             ACCOUNTS = json.load(fp)             for temp in ACCOUNTS:                 if temp["id"] == uid:                     temp["dollar"]+=money                     break             with open(ACCO_FILE, 'w') as wfp:                 json.dump(ACCOUNTS, wfp)         lock.release()         return new, 201     else:         return {"error": "Request must be JSON"}, 415</pre>	<pre>new_dict = {} new_dict["uid"] = user["id"] new_dict["money"] = money response = requests.put(URL_ACCOUNT, json=new_dict) print(response.status_code) print(response.headers) print(response.text) MessageBox(0,"You "+str(money)+" dollars", "Yahtzee",MB_OK)</pre>



配對:當使用者在大廳中點擊 Multiple mode，就會開始進入配對，Client 會開啟一 Thread 來傳送 request 到 Sever 要求加入配對，Server 會先查看該玩家是否正在進行遊戲，或者已經在配對，若有則回傳告知，如果該玩家未在配隊或是在遊戲中，就會加入隊伍，並將該 thread 透過 sign 的方式鎖起來，不斷去等待新開的房間。30 秒後就會以四人一組的方式分配，若有少人就補電腦。

Server	Client
<pre>def join():     global join_q     global player_inform     uid = request.args.get('uid')     #{uid:[playid,room]}     if uid in player_inform.keys():         return {"error": "You are playing"}     if (join_q.empty()):         t = threading.Thread(target=waiting)         t.start()     if uid in list(join_q.queue):         return {"error": "You are waiting"}     join_q.put(uid)     print("Player "+uid+" join the Queue")     while True:         if player_inform.get(uid)!=None:             if len(room)-1&gt;=player_inform[uid][1]:                 print(player_inform)                 return jsonify(player_inform)</pre>	<pre>global player_state global Multiple_sign my_params = {} my_params["uid"] = user["id"] response = requests.post(URL_JOIN, params = my_params) print(response.status_code) print(response.headers) player_state = response.json() # response if "You are waiting" in str(player_state):     Multiple_sign=False     self.send=True     MessageBox(0,"You are Waiting", "Yahtzee",MB_OK)     return elif "You are playing" in str(player_state):     Multiple_sign=False     self.send=True     MessageBox(0,"You are Playing", "Yahtzee",MB_OK)     return print("player_state:"+str(player_state)) self.connected=True</pre>

實際配對程式碼:

```
def waiting():
    print("startwait")
    time.sleep(20)
    global join_q
    global player_inform
    while not join_q.empty():
        print("in")
        player=[]
        for i in range(4):
            if not join_q.empty():
                #{uid:[playid,room]}
                uid=join_q.get()
                player.append(uid)
                player_inform[uid]=[i,len(room)]
                print("playerid: "+str(i)+" room number: "+str(len(room)))
            else:
                print("robot")
                continue
        print(len(player))
        if len(player)==1:
            room.append(main_dice('4','#'+123))
        elif len(player)==2:
            room.append(main_dice('4','#'+23))
        elif len(player)==3:
            room.append(main_dice('4','#'+3))
        elif len(player)==4:
            room.append(main_dice('4','#'))
        room[len(room)-1].start()
    print("endwait")
    #return server\連上Queue 結束waiting lobby
```

載入資料:回傳遊戲進程給 Client，Client 端每 0.5 秒會傳送自己的 iD 和 Client 已讀取到的 index，然後 Server 對需要的部分進行擷取，即可避免 Queue 因為被其他玩家存取後資料損失的問題。

```
def get_game():
    global player_inform
    new = request.get_json()
    uid=new["uid"]
    queue=list(room[player_inform[str(uid)][1]].q_out_total.queue)#
    result=[]
    for i in range(new["font"],len(queue)):
        result.append(queue[i])
    return jsonify(result)
```

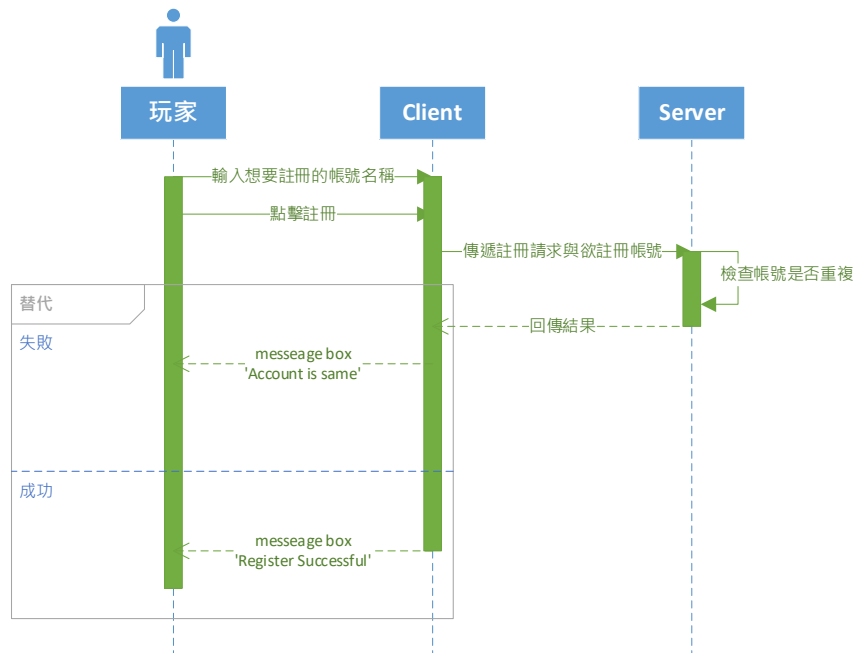
資料輸入:在玩家的回合時，可以傳送訊息給 Server，且如下圖傳送

@@+userid 時可以直接將該玩家的位子替換成電腦。

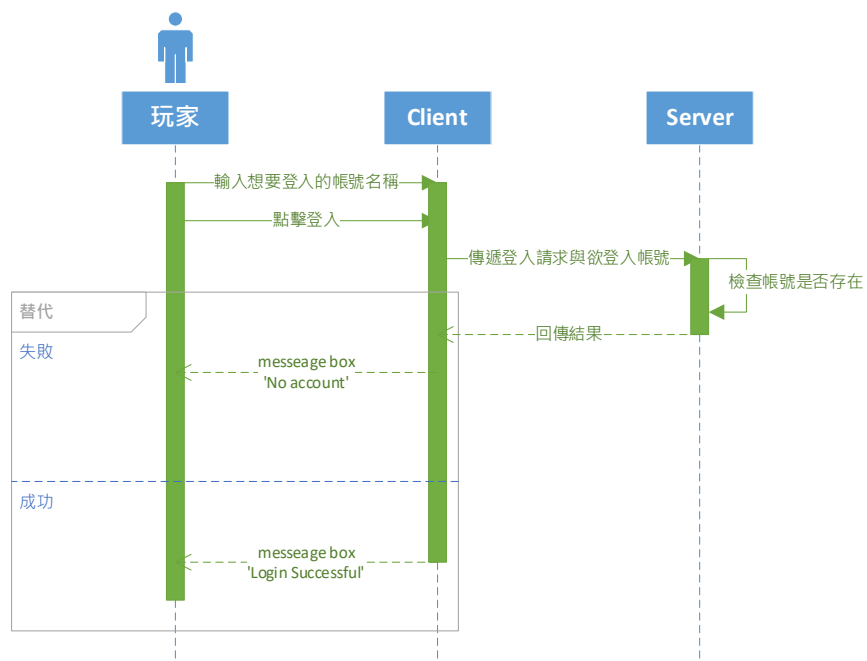
```
my_params={}
my_params["input"]="@"+str(player_state[str(user["id"])[0]+1)
my_params["uid"]=user["id"]
response = requests.put(URL_INPUT, json = my_params)
```

## 循序圖

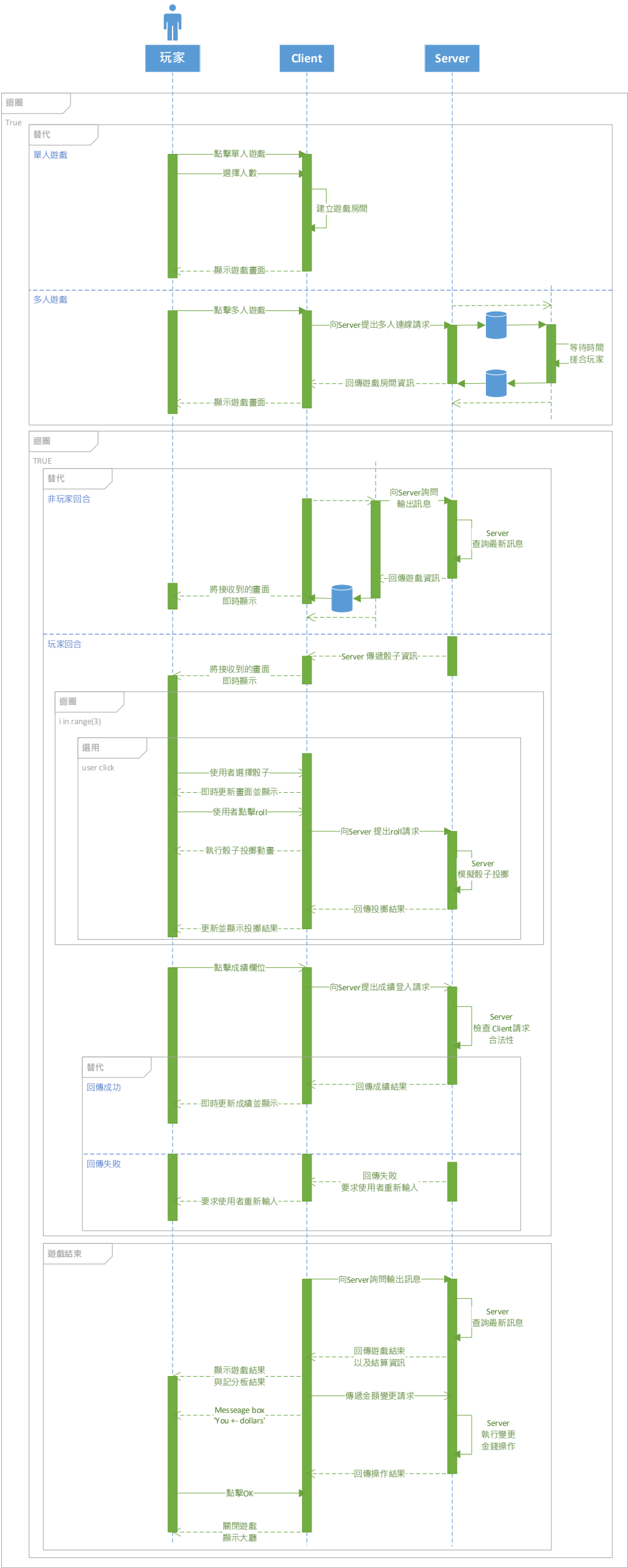
- 使用者註冊



- 使用者登入



● 遊戲過程



## 成果：

那我們一進到遊戲就會進入登入大廳，這邊採用賭俠作為我們的主題，強調需要有 300 萬美元才能參加遊戲，是我們的主題展現，如果連 300 萬美元都沒有就會被逐出遊戲，沒有重來的機會，下方為登入少於 300 萬美元帳號的畫面。



進入大廳後，可以在左上方看到自己的帳號名稱，右邊則為所持有的資產，使用者在這邊需要選擇自己想要以哪種方式進行遊玩。



點擊 Local games 就可以選擇有幾位玩家要同時遊玩，可以選擇 1~4 名玩家同時在本地遊玩。

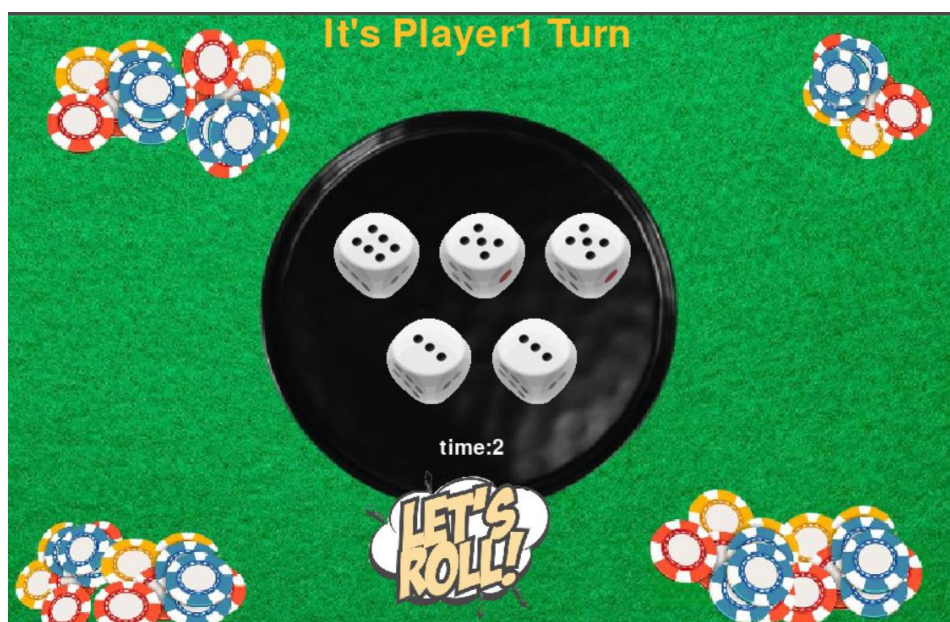




若點擊 Multiple game 就會開始進行配對，開啟一新 Thread 來等候配對成功，但配對成功就會進入遊戲主畫面。



遊戲主畫面 UI 外部設計，骰子固定置中，且具有骰子滾動的畫面，點擊 Let' s ROLL 按鈕來達到重骰，上方也有顯示該回合是否是自己的回合以及當前重骰機會次數的計算。



畫面的右方為計分表，用來表示目前所得分數，以及確定要選擇得計分方式，計分方式不可重複選取，可以點擊自己位子上的 select 來確定計分。

UPPER SECTION		HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4
Aces	● = 1	Count and Add Only Aces	select			
Twos	● = 2	Count and Add Only Twos	select			
Threes	● = 3	Count and Add Only Threes	select			
Fours	● = 4	Count and Add Only Fours	select			
Fives	● = 5	Count and Add Only Fives	select			
Sixes	● = 6	Count and Add Only Sixes	select			
TOTAL SCORE		→				
BONUS <small>(if total score is 63 or over)</small>		SCORE 35				
TOTAL <small>Of Upper Section</small>		→				
LOWER SECTION						
3 of a kind		Add Total Of All Dice	select			
4 of a kind		Add Total Of All Dice	select			
Full House		SCORE 25	select			
Sm. Straight <small>Sequence of 4</small>		SCORE 30	select			
Lg. Straight <small>Sequence of 5</small>		SCORE 40	select			
YAHTZEE <small>5 of a kind</small>		SCORE 50	select			
Chance		Score Total Of All 5 Dice				
YAHTZEE BONUS		✓ FOR EACH BONUS SCORE 100 PER ✓				
TOTAL <small>Of Lower section</small>		→				
TOTAL <small>Of Upper Section</small>		→				
GRAND TOTAL		→				

投擲的過程中可以鎖定骰子，以保留想要的點數，來達成更高難度的得分方式。





該回合結束後，螢幕上方出現 Not your Turn!!，表示 USER 需等待他人的回合，之後才輪到自己進行操作。



遊戲結束後就會顯示名次及得名，並彈出視窗告知自己的金額被更動，點及確定後就會再次回到大廳。

Not your Turn!!

1 st is player3 get:264 point. earn\$600!  
2 nd is player1 get:233 point. earn\$300!  
3 rd is player2 get:204 point. earn\$300!  
4 th is player0 get:202 point. earn\$600!

Yahtzee  
You -600 dollars  
確定

UPPER SECTION		HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4
Aces	● = 1	Count and Add Only Aces	select	4	3	2
Twos	● = 2	Count and Add Only Twos	8	6	4	6
Threes	● = 3	Count and Add Only Threes	6	12	3	9
Fours	● = 4	Count and Add Only Fours	4	4	6	6
Fives	● = 5	Count and Add Only Fives	15	10	10	15
Sixes	● = 6	Count and Add Only Sixes	12	24	12	18
TOTAL SCORE						
BONUS	1 star score is 63 or over	SCORE 35				
TOTAL						
LOWER SECTION						
3 of a kind	Add Total Of All Dice	27	25	21	21	
4 of a kind	Add Total Of All Dice	16	27	25	18	
Full House	SCORE 25	25	25	25	25	
Sm. Straight	Sequence of 4	SCORE 30	30	30	30	
Lg. Straight	Sequence of 5	SCORE 40	40	40	40	
YAHTZEE	5 of a kind	SCORE 50	select	select	select	
Chance	Score Total Of All 5 Dice	19	24	23	22	
YAHTZEE BONUS	FOR EACH BONUS SCORE 100 PER					
TOTAL	Of Lower Section		169	195	176	224
TOTAL	Of Upper Section		33	38	28	40
GRAND TOTAL			202	233	204	264

## 結論：

在這份專題中我們使用了 Remote Procedure Call(RPC)，將本地程式的 function 或是 Object 裡的 method 獨立在伺服器上，利用 TCP/HTTP 各種網路通訊方法，使其他使用者可以在自己的裝置上面運行，並可以透過網路呼叫存在遠方的 function 或是 method，來達到我們連線互動的功能，在初期我們嘗試使用 socket 來進行連線，可其發送封包時過於繁瑣複雜，容易導致程式的錯誤率提高，且管理 socket 也是需要大量的時間設計，所以後來我們選擇使用 RPC 後便可以達到更有效率的 Client and Server 的互動，將時間更多的發揮在遊戲本身上的設計，況且在處理有關於伺服器掌管的伺服器資料庫時，RPC 所提供的專屬程式（查詢、更新等）可以更有條的讓遠端電腦（或客戶端）呼叫，使 Server 能提供各種服務程式，讓遠端電腦透過網路來查詢，大大減少了資料間傳輸開發的過程。這堂課伴隨著大量的網路程式的各種理論與基礎，在這份專題中，我們努力的呈現出課堂中所有的知識來達成我們的目的，這以不只是一份課堂專題，而是真正帶的走的硬實力！

## 參考資料、參考文獻：

<http://zhuzhuodong.com/tech/android/wx-touzi-img-source>

參閱時間: 10/03

[https://blog.csdn.net/weixin\\_46373994/article/details/104441657](https://blog.csdn.net/weixin_46373994/article/details/104441657)

參閱時間: 10/11

<https://www.pygame.org/docs/>

參閱時間:10/13

<https://docs.python.org/zh-tw/3/tutorial/index.html>

參閱時間:10/15

Kathiravelu P., Sarker F. - Python Network Programming Cookbook,

Second Edition – 2017,

<http://zenhadi.lecturer.pens.ac.id/kuliah/WorkshopPemrogramanJaringan/Kathiravelu%20P.,%20Sarker%20F.%20-%20Python%20Network%20Programming%20Cookbook,%20Second%20Edition%20-%202017.pdf>

TJLYU 2022 Network Programming

<https://zh.m.wikipedia.org/zh-tw/%E8%B5%8C%E4%BE%A0>

<https://zh.wikipedia.org/wiki/%E5%BF%AB%E8%89%87%E9%AA%B0%E5%AD%90>