CS3025301:Software Engineering



結案報告 for 決戰俄羅斯

C++ & Qt 實作─俄羅斯方塊

Team: 第17組

Team members: B10415043 陳弘展、B10415014湯志超、B10415010 張竣東、B10415042 蔡奇麟、B10401015林恆毅

Contents

1. Introduction 3
2. Design Considerations 3
   1. Assumptions 3
   2. Constraints 3
   3. System Environment 3
3. Architectural (High-Level) Design 3
   1. Overview 4
   2. Rationale 4
   3. Conceptual (or Logical) View 5
4. Low Level Design 6
   1. Class Diagram 6
   2. Sequence Diagram 7
   3. Activity Diagram 10
   4. State Diagram 12
5. User Interface Design 13
6. Experiences 18
   1. Problems we’ve encountered 18
   2. Review of MVC 18
7. **Source Code 20**

# Introduction

本篇專題報告主題為以 C++ & Qt 實作俄羅斯方塊，並且使用我們在軟體工程裡面學到的MVC架構來實作。

所謂的『俄羅斯方塊』是由幾種四格骨牌構成，每種都由四個方塊組成。開始時，一個隨機的方塊會從區域上方開始緩慢繼續落下。落下期間，玩家可以以90度為單位旋轉方塊，以格子為單位左右移動方塊，或讓方塊加速落下。當方塊下落到區域最下方或著落到其他方塊上無法再向下移動時，就會固定在該處，然後一個新的隨機的方塊會出現在區域上方開始落下。當區域中某一橫行（列）的格子全部由方塊填滿時，則該列會被消除並成為玩家的得分。玩家在遊戲中的目的就是盡量得分。當固定的方塊堆到區域最頂端而無法消除層數時，遊戲就會結束。

# Design Considerations

## Assumptions

1. 程式為離線執行，遊戲結束後不儲存遊戲資料(例如最高分數)。因此一切操作都是都in-memory的。

2. 操作以鍵盤及滑鼠進行輸入。

## Constraints

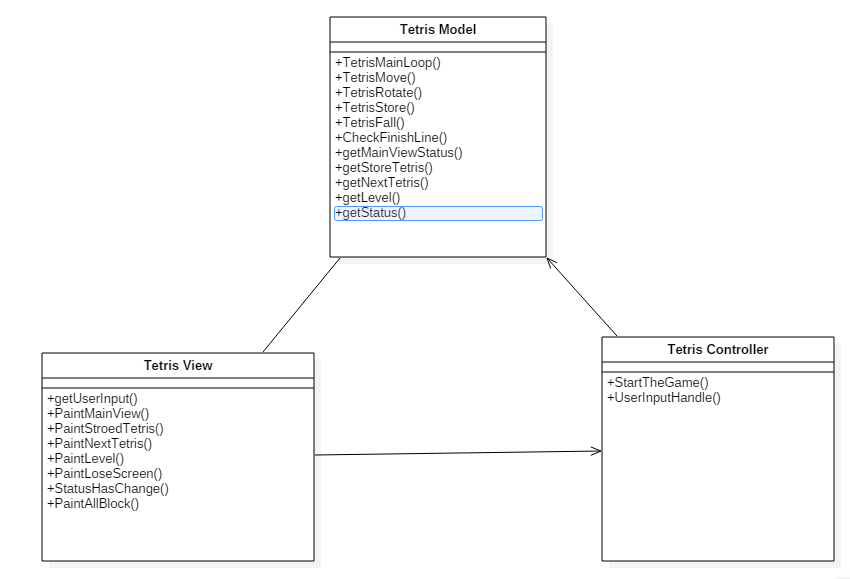
必須要能方便替換介面，準確地說，是要能替換一個object就能更換介面，為此我們採用了MVC架構。

## System Environment

執行環境要求為X86架構下的Windows 7及Windows 10平台。

# Architectural (High-Level) Design

## Overview



MVC:

TetrisMode: A class that manages the Tetris entries and captures the state of the application. Whenever the state is changed, it notifies Tetris, which should then refresh the user interface based on the state of the application.

TetrisView: A class that manages the graphical or textual interface to the user based on the state of the application, and notifies Tetris controller when an input is received.

TetrisController: A class that controls the operation of the entire application. It changes the model state of the application and updates the data model based on user input.

## Rationale

以習慣代替配置：

在多人開發的專案中常常因為大家對於專案架構的不了解而造成程式碼混亂，檔案放在不該放的資料夾內是常有的事情，但在MVC當中已經明確規範了很多規則，例如Controller的檔名必須是Controller結尾，View一定要放在專案中的Views資料夾底下，所以在開發MVC之前不需要再特別的去跟同事互相討論專案的一些細節規範也能夠輕鬆的擁有一個MVC架構的專案，並且在接手其他MVC案子的時候可以比較容易找到程式存放的位置，不過在這之前也是需要先行了解MVC的定義好的開發習慣或是團隊間在自行加強所需要的規則，不然也是免不了會有亂放的情況。

關注點分離：

簡單來說就是一次只關注在一件事情上，舉裡來說我們的團隊中就很明確的細分出誰負責Model的開發、誰負責View、誰負責Controller，各自將功能封裝起來之後就可以只專注在自己負責的事情上面，因為大家擅長的部份也都不一樣，像我資料庫那塊就弱到爆，雖然不適合寫Model但只要能使用別人寫好的方法我就能夠去處理其他我能做的事情，所以一次只關注在一件事情上就很適合我這種不是全能型的選手，或是能把問題細分出來再慢慢各自處理會會比同時一次性的處理掉所有事情來得簡單一點。另外遇到錯誤時怎麼辦？過去在php裡面那種義大利麵的寫法在遇到bug時不易找出錯誤的地方，但透過關注點分離很容易可以找到該負責的是哪一隊友。

分層架構：

除了預設的MVC的架構外也能自己在切割出更多功能層，讓專案維護性提高，例如可以把一些商業邏輯在個別抽出來，簡化程式碼或提高重複利用性，例如把Model定義好之後，那麼在同一個方案中前台與後台的專案就可以共用同一個Model(設成類別庫)。

## Conceptual (or Logical) View

參照3.1的圖進行進一步解釋

Tetris Controller:  
 start the game():開始遊戲

UserInputHandle():處理user的key

TetrisModel:

TetrisMainLoop(): 遊戲主要流程操控，會每N(N可動)秒執行一次。

TetrisMove(): 左右移動

TetrisRotate(): 順/逆時鐘旋轉

TetrisStore(): hold currentSqure,currentSquare=nextSquare

TetrisFall(): 讓使用者將當前方塊直接落下至最底方

checkFinishLine(): 判斷是否有任何一行已經完全被填滿

TetrisView:

paintAllBlock(): 繪製主遊戲UI

paintMainView():繪製UI中的主要遊戲介面內的內容

PaintStoredTetris():繪製UI中的storedSquare

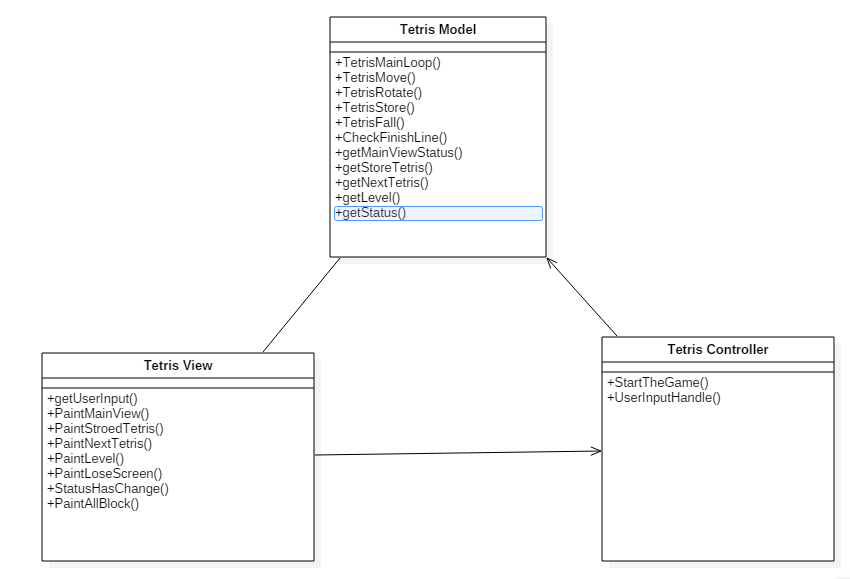
PaintNextTetris():繪製UI中的nextTetris

PaintLevel():繪製UI中的level

PaintLoseScreen():繪製結束遊戲畫面

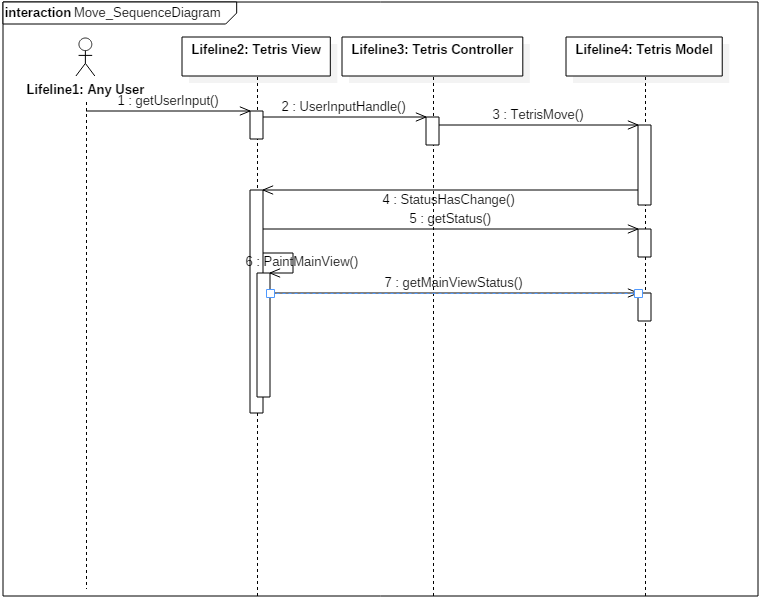
# Low Level Design

## Class Diagram

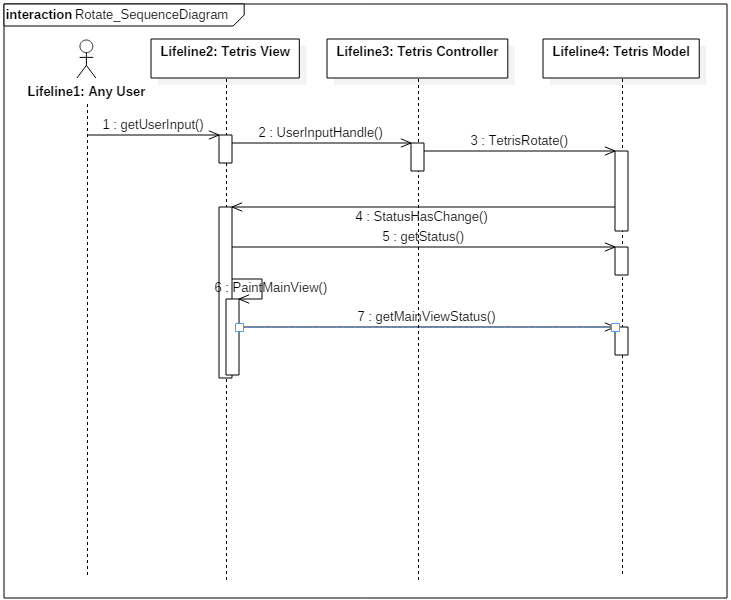


## Sequence Diagram

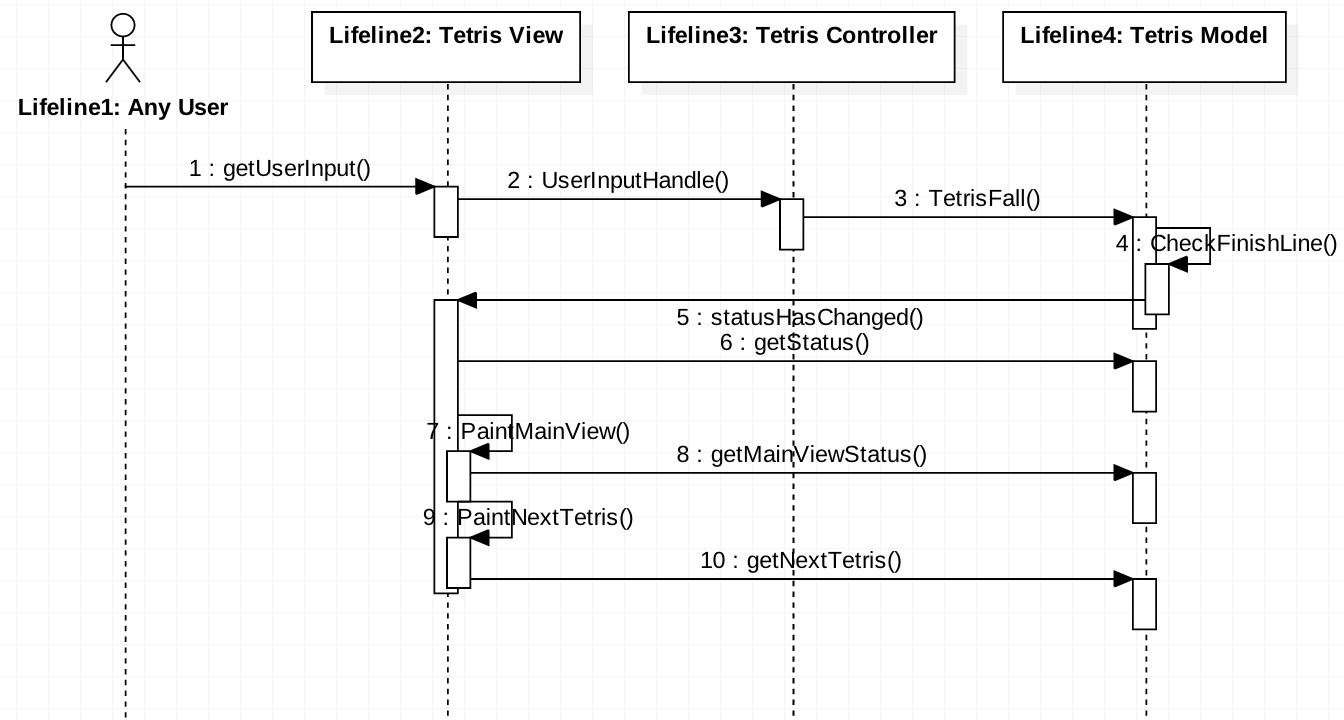
**Tetris Move：**

****

**Tetris Rotate：**

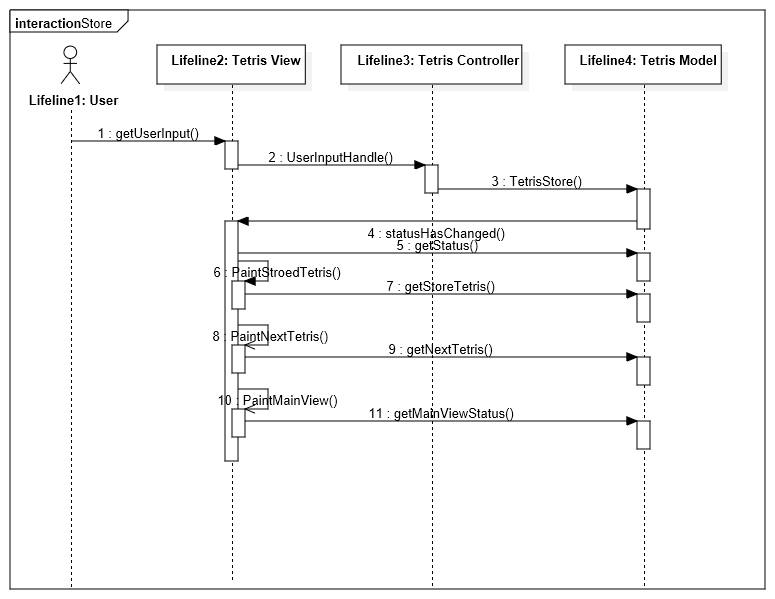
****

**Tetris Fall：**

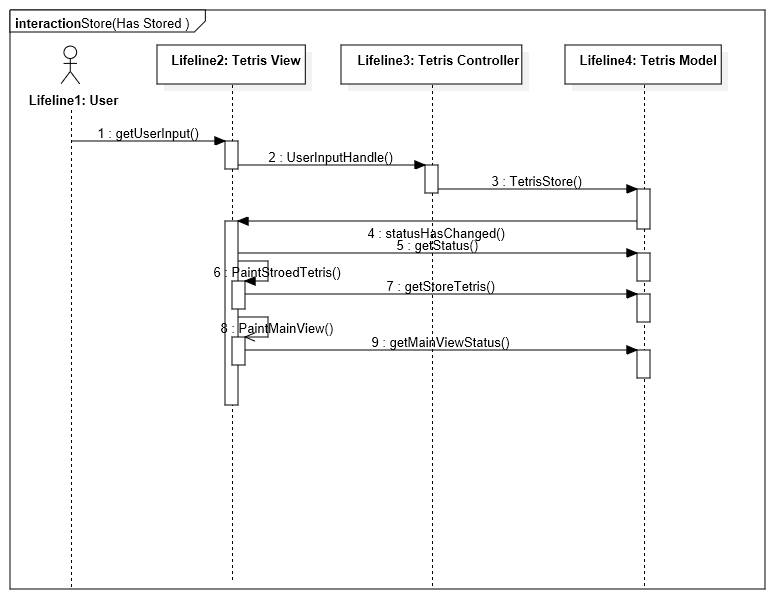


**Tetris Store：**

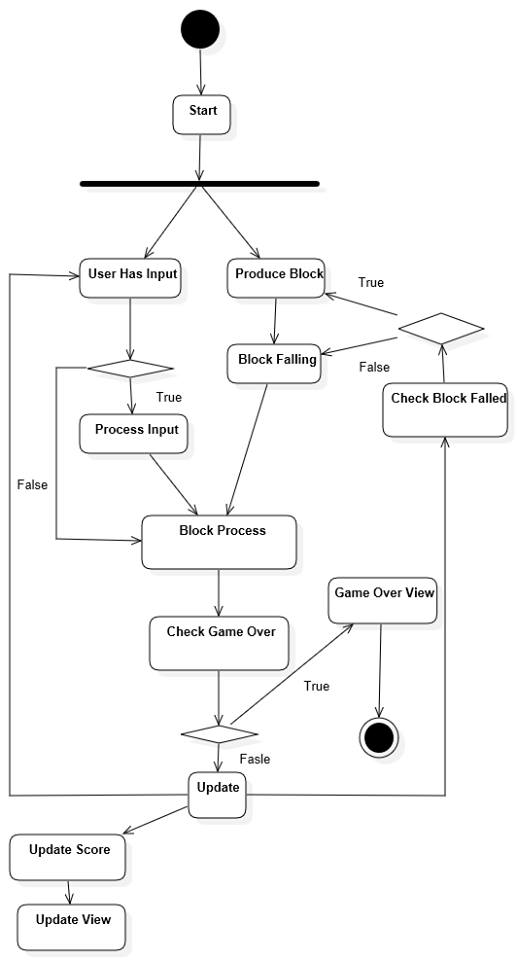
起始沒有東西在儲存裡：



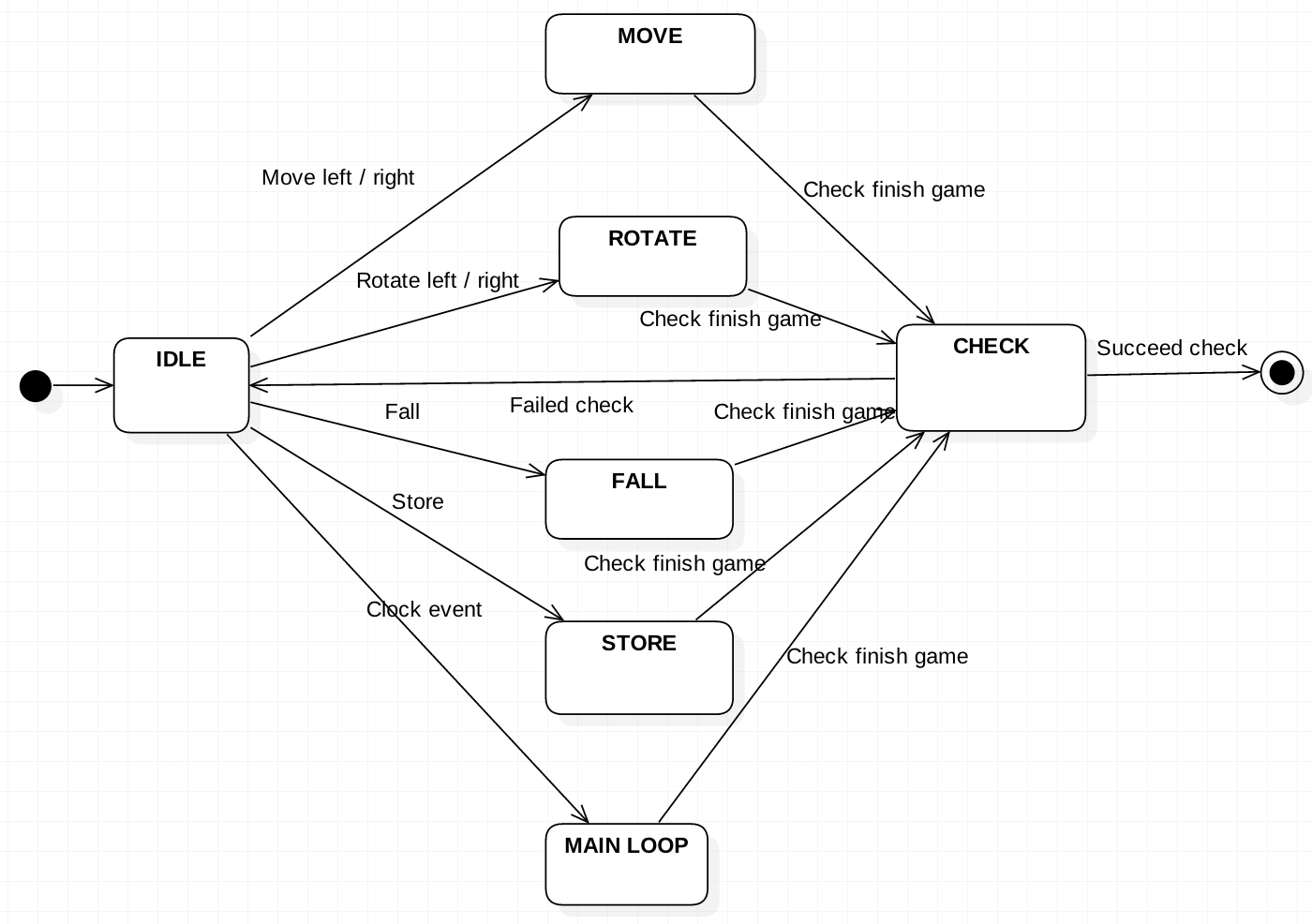
有東西在儲存裡：



## Activity Diagram

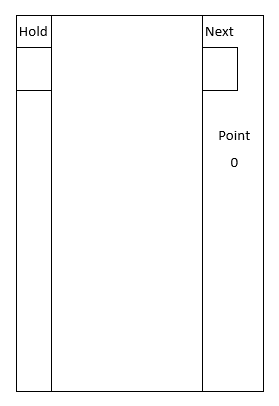


## State Diagram

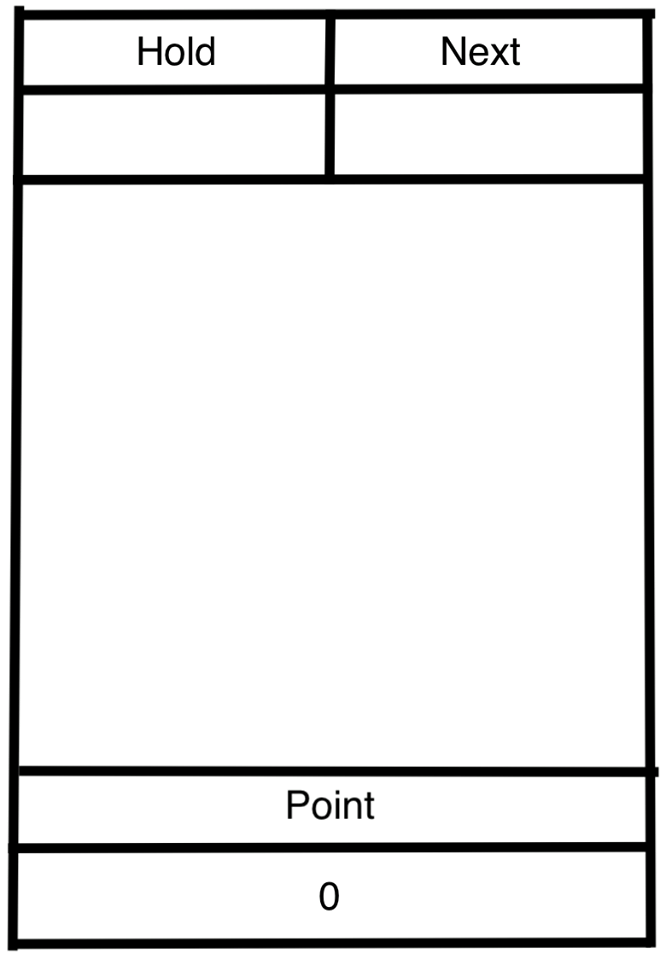


# User Interface Design

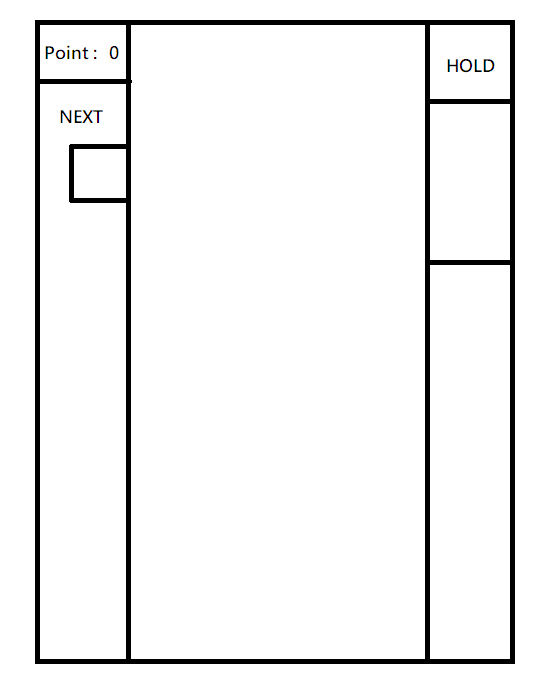
陳弘展：



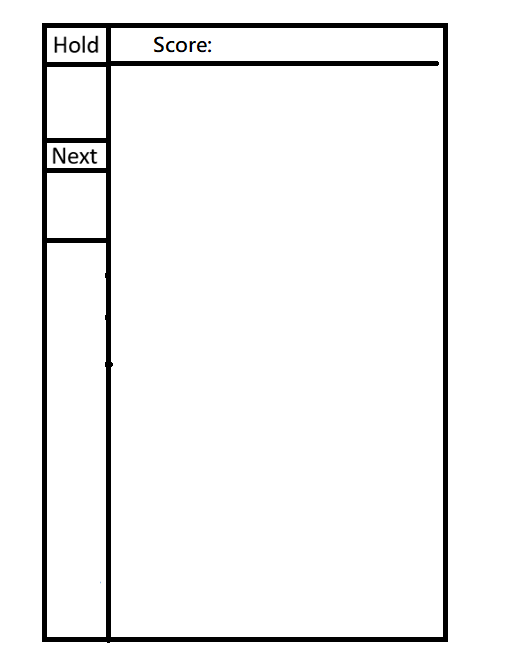
林恆毅：



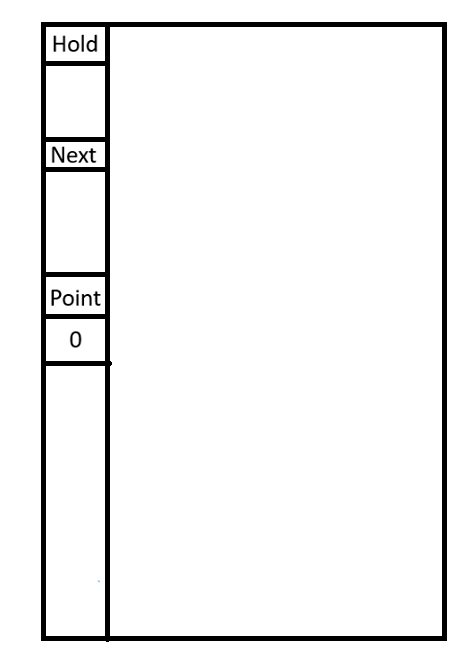
張俊東：



湯志超：



蔡奇霖：

**

# *Experiences*

## *Problems we’ve encountered*

*1.*如果遇到正在計算玩家指令的時候遇到*timer*觸發的話，該怎麼處理*?* 當有人按下右移鍵，系統正在運算時，結果*timer*時間到處發，那應該等這次又移指令做完才做*timer*觸發還是要忽略這次*timer*觸發*?*

*2.*如果玩家在進行動作時方塊會剛好落在主體上，那應該判斷落地並開始新的一個方塊還是要讓玩家可以繼續移動直到下一次*timer*處發才判斷是否落地

*3.* 旋轉問題

## *Review of MVC*

經過我們這次一起用MVC合作完成這個專案後，我們發現MVC有許多優點，如：習慣代替配置、關注點分離、分層架構、開放原始碼、等！MVC模式將系統切割成三個元件：Model包含核心功能與資料、View顯示資訊、Controller處理使用者的輸入。View和Controller合在一起構成使用者介面，並透過通知機制確保使用者介面與model維持一致性。

**Model**封裝核心資料與功能，並且獨立於顯示方式與輸入行為。**View**負責顯示**Model**的資料，一個**Model**可以包含多個**View**。每一個**View**有一個**Controller**用來接收輸入，並將其轉換成對**Model**或**View**的呼叫。

區分Model與View/Controller讓我們可以使用多個View來顯示相同Model。如果使用者透過某個View的Controller改變Model的資料，所有其他相依於該Model的View將會收到Model關於資料異動的通知，如此一來這些View便可從Model接收異動的資料並用來更新自己所顯示的資訊。

一些我們用過MVC的心得:

• 可以使用多個View來顯示相同的Model。

• 顯示相同Model的多個View之間可以保持同步。

• 可插拔的View和Controller。因為View/Controller和Model分離，所以可以改變用來顯示Model的使用者介面，甚至可以在runtime做出這樣的改變。

• 當系統轉移到其他平台的時候，可以只更換使用者介面而繼續保留Model。

• MVC可以被使用於應用程式框，減化互動程式的開發。

• 增加複雜度。嚴格區分MVC有時候會增加系統複雜度，例如在menu或簡單的text元件上套用MVC可能就變成殺雞用牛刀。

• 雖然MVC區分了Model、View、Controller，但View和Controller其實是緊密結合的元件，很難各別被重複使用。

• View/Controller和Model之間有緊密結合的關係，如果Model改變很可能會造成View/Controller也跟著改變。

• 更新View所造成的低效率資料存取。為了更新資料，View可能需要呼叫Model好幾次以獲得所需資料。不必要的讀取未異動資料將降低執行效能。