**Software Maintenance Document for Master’s Project**

Chen Gao, MIT

Flinders University

School of Computer Science, Engineering, and Mathematics in the Faculty of Science and Engineering

Aim

The aim of this document is providing an overview of my coding to successors who may keep exploring player behaviour prediction method in future.

Constrains

All information contained in this document is based on project files submitted to git-hub at 24 Aug 2015, reference number 33f5d5daf81c3c0186bf70ee3f61404c6b72a68, code can be browsed at following address:

https://github.com/maicallist/MasterThesisProject/commit/33f5d5daf81c3c0186bf70ee3f61404c6b72a689

Setup Environment

Project was built on Microsoft XNA 4.0 framework and Microsoft Visual Studio 2013. Install VS2013 XNA 4.0 Refresh plugin, if program indicates error, try installing XNA 4.0 (not 4.0 refresh) again, installation may fail, but then program should work.

OS: Windows 7 SP1

Code Overview

The entire game framework was borrowed from one of my previous game. It handles player input (InputManager class), how game screens transit (ScreenManager), animation effect (FadeAnimation) and so on. All master project related code are in two class: DataCollector (single player mode) and GamePlayScreen (multiplayer mode).

DataCollector Class

In single player mode, we try to collect various data to create prediction models. Players have following controls:

Left arrow key and right arrow key – move paddle at bottom of screen

Space – start or restart game

XNA has two major methods to control the game, update() for value changing and draw() for rendering objects. We have set the game update 60 times each second.

XNA framework keeps tracking elapsed game time, therefore, in update() line 223 (@223), we updates current time tag every time we call update(), if in current cycle of update, we successfully collect some data, store them with current time tag.

Recording Prediction Data

In update(), we call doPrediction() to check if current game state fits our prediction model. @1020-1083, we first calculate the distance between the ball and the paddle, then send some parameters to predict\_disToPlatform() method which returns a value that model believes the should be at or around such height on screen. pre\_disToPlatform method has four overload types corresponds to four prediction models @1086-1154. When doPrediction() receives results from predict\_disToPlatform(), it first rules out some results that rarely happens in real gameplay (height < 100 which no one can react so fast, height < 550 which player react that late usually results in game over), then check if the ball’s current vertical position is near our prediction (by near we mean if the residual between the ball’s height and our prediction is less or equal then 3, we call that it’s a match. This value 3 was tested by many experiments along with other potential values, and we believe it’s the best value to keep the balance between prediction accuracy and prediction match rate).

AI and Computer Paddle Controls

Computer paddle is controlled one method only – MoveComPlatform() @889. This method can move com paddle to its destination position. It requires two parameters: destination and flag.

Destination is the horizontal position where computer paddle is going to collide with the ball and is calculated by method CalcComPlatformPosition() @821. CalcComPlatformPosition() is called in two places, every time when the ball collides with player paddle and when we change the ball’s velocity. Except calculating com paddle collision position, CalcComPlatformPosition also request to move computer paddle by flags up movePlatformCom @845. In Update(), if flag movePlatformCom is up @454, we then further check current game difficulty level. Moving paddle to the desirable position based on different AI of each level. Flag down movePlatformCom when paddle is at correct position, waiting for next time CalcComPlatformPosition() is called.

For our easiest level “Hard”, we simply move paddle to collision position when CalcComPlatformPosition() request to move the paddle.

For level “VeryHard”, we apply prediction models to AI, paddle only moves when flag movePlatformCom is up and flag veryhardMove is up. veryhardMove is up when we have current prediction result matches the ball’s vertical position @1157-1188. After the ball is bounced back, we move paddle back to centre of screen based on prediction model @1212-1228.

For level “ExtremeHard”, we try to deceive player recognizing AI and real human player by making AI randomly moves to wrong direction and sometime miss the ball. In Update() under ExtremeHard level, after we call CalcComPlatformPosition() @ 332 works out collision position, we call CalcComWrongPosition() @334 which works out how far computer paddle can move on wrong direction that is enough to cause the paddle miss the ball, then flag up moveComPlatformWrong @1008. In Update(), we first move computer paddle to wrong direction, and get a random number from number generator, if the number is in range, we keep moving paddle on wrong direction, until it reaches the position we work out from CalcComWrongPosition(), then we move paddle to the right collision position. In this case, if the paddle reaches the furthest position on wrong direction, it means the paddle certainly will miss the ball. But if we get a random number out of range, then we stop moving on wrong direction, and move the paddle to correct collision position. This range is controllable @484, we currently set that on 95 precent of chance the paddle will keep moving on wrong direction.

Flag is anther parameter we need to move computer paddle through MoveComPlatform(). It specifies what operation we are doing: moving paddle to collision position (flag: 1); to a wrong position on wrong direction (flag: 2); to collision position based on prediction model (flag: 3); to centre of screen after collision (flag: 4). Paddle only moves when one of these flag is up, and the flag is down when paddle is at correct position @894-908.

GamePlayScreen Class