COMP4521 Mobile Application Development

Final Report

Project topic: A Real-Time Strategy Game Assisting App

Group name: Group AF

Group member:

Ding Ziheng (20677122) (Leader)

Word count: **2859 word.**

Date: **13/05/2023**

1. Introduction

1.1 Overview

Real-time strategy (RTS) is a subgenre of strategy video games that each participant plays simultaneously instead of taking turns. Usually, the participant plays as a commander of an army, collects resource on the map, positions structures and buildings, maneuvers multiple

units and destroys their opponents' assets to win the game.[1] Typical examples of RTS games are Red Alert, Age of Empire, StarCraft and etc, which are well-known and popular in the last two decades.

However, RTS is always considered as the hardest type of computer game. Taking StarCraft II as an example, it has hundreds of commands and hotkeys which are used intensively in gameplay [2], making them extremely difficult to be remembered or to be performed. To make it worse, RTS games are usually fast paced. In StarCraft II, players` performance is measured by action per minute (APM), standing for how many keys strikes a player makes in a minute. Usually in a match, the higher APM one player has, the faster one is. E-athletes have 400+ APM, and players in Grand Master League, which is the highest rank in the game, usually have an APM around 200 to 300. To overperform others and win the game, player must practice enormously, driving the game away from its original purpose of being relaxing and enjoyable.

1.2 Market survey

In the market research I conducted, I surveyed on 6 StarCraft II players. They are all relatively old players with over 3-year experience on the game. However, four of them still have an APM below 120 in their average game play, and even for the best game they have ever played, only one interviewee`s APM reached 180. All of them reflected that they felt micro controlling units difficult for most of the times. More specifically, among the seven micro tasks being surveyed (Q5 of market survey, Appendix), the first two micros require fast but only repetitive actions, which were generally considered doable. However, when it comes to the third to fifth tasks, which require much more precise control, or the last two tasks which need player to quickly switch and give command to armies on different locations, they felt extremely hard to do so.

The above survey result confirms two fundamental needs of RTS micro control:

1. Precision
2. Fast action

Also, it mined out a new problem that many players struggling with:

3. Rapidly switching from place to place on the map.

1.3 Objectives

With the hope of improving RTS gameplay experience, this project aims at redesigning the game control mechanism.

1. Using smartphone to control the gameplay.
2. Design a mapping between gestures input and keyboard and mouse input.

With this project, players can give any commands by a few taps and gestures rather than a sequence of complex key strikes, making their gameplays easier and more intuitive. This project will focus on the most well-known RTS game, StarCraft II to examine the viability of the idea.

2. Methodology

2.1 Application Structure

The project is based on the workflow below.

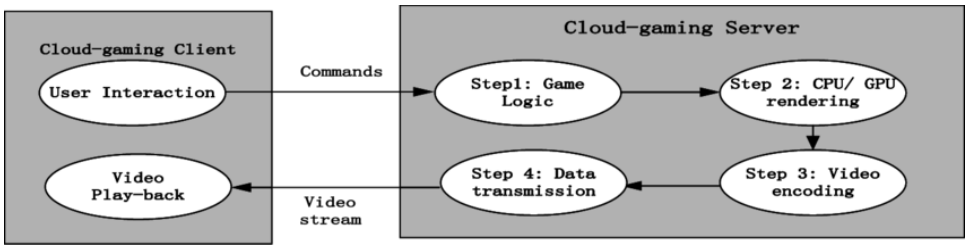

Figure 1: Workflow of cloud gaming app

The app functions as the following:

1. The local machine takes the place of the server end and hosts the game.
2. The game screen is mirrored to the smartphone on real time.
3. The smartphone is connected to PC via cable, act as the input device. Once any gesture is captured, the smartphone end will request the PC to carryout corresponding action.
4. The local machine calls APIs in the operating system to imitate keyboard and mouse to perform the action in the game.

2.1.1 Smartphone end

The smartphone end provides the user interface of this system. It has an activity to display the real time screen of the gameplay and handle gesture inputs. It will be implemented by a set of rules to determine what action the player intends to conduct in game when a certain gesture is

recognized. Then, it will send the player`s intent with the PC, instructing the PC to perform the action.

2.1.2 PC end

The PC end plays the rule of hardware drive for the smartphone end and provide software interface for the smartphone to access the game. Once being connected to the smartphone end, it sets up communication with the smartphone end, resolves what action it should perform according to the message received from the phone, and carries the actions out in game.

Also, the PC end provides computational resource for the system. As PC games usually requires several giga bytes memory and storage in nowadays, it is not viable to set up a virtual Windows or Mac machine to run those games on the smartphone. Thus, the game will still be hosted on the PC and the smartphone only takes control of the game.

2.1.3 Communication

The communication part has two channels: video stream channel and commands channel. The video stream channel is for mirroring the game screen from the PC to the smartphone on real time, as the user need to know what is being on in the game. The commands channel is for the smartphone end to transmit the result of gesture recognition to the PC end. The message will be encoded at the smartphone end and decoded at the PC end for more efficient processing.

As the app does not need any processing on the screen image, or any feedback message of the gesture commands, both channels are designed to be one-way communication.

2.2 Features realized

Every command in game is redesigned and mapped to gestures on the touch screen. The game interface can be divided in to seven regions as shown in the figure and commands are redesigned as the following:

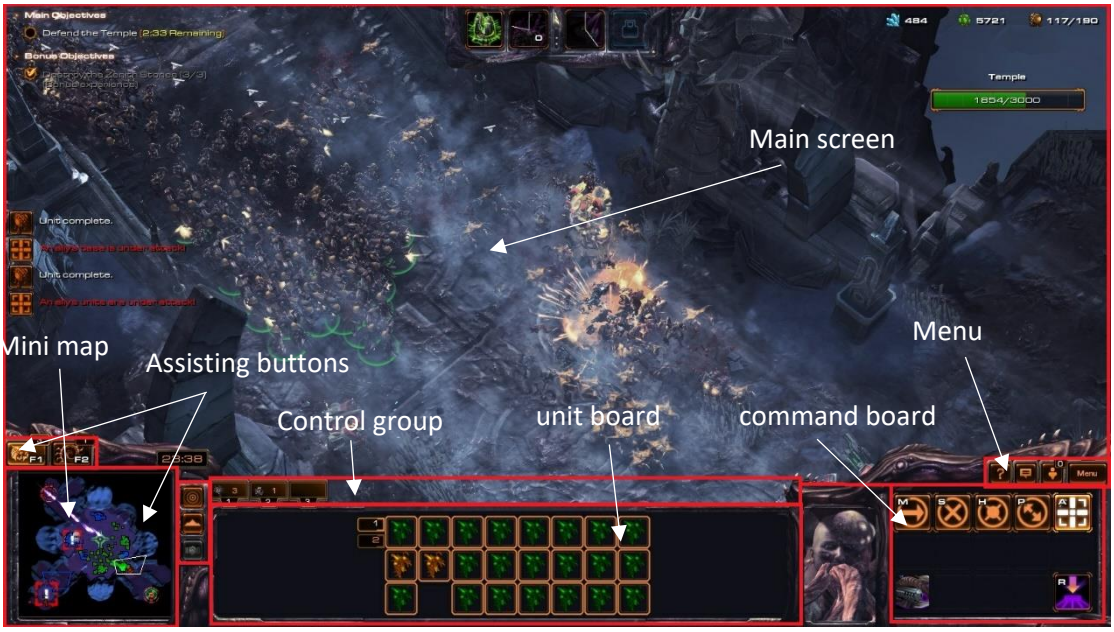

Figure 2: Sections of gaming interface

This in game screen is displayed on the smartphone, exactly same with the PC screen

2.2.1 Main screen

Main screen is where the player can see and control their army. The most frequently used game commands on the main screen are ***select*** and ***move***. Originally, players left click on a unit to select it, or drag a square region to select a bunch of units, then right click on the ground to move the selected army. Moving commands has an alternative variation called ***attack move*** or ***A move***. Player first hit the key ***A***, the hotkey of attack command, then left click on the ground so that their army will march to the location and automatically attack into any enemies in their way.

The main screen only displays a part of the map. Players can use arrow keys to scroll their screen and hover over the map.

These commands are redesigned to the following gestures on screen:

1. A single tapping on the main screen area on the smartphone triggers a ***move*** command.
2. A double tapping is treated as an ***attack move***.
3. Long pressing on screen as the left click for ***select*** single unit.
4. Zooming in gesture for ***select*** a region.
5. Dragging on screen to ***move the camera***.

2.2.2 Mini map

The mini map shows the entire map but with much less details than the main screen. There is a white quadrilateral area marking the current location of the camera. Left

clicking on the mini map will move the camera to that location immediately, and on the smartphone, players can do so by simply tapping on the minimap

2.2.3 Command board

Command board displays icons of commands that a unit can receive. It includes universal commands such as stop and patrol, or unit specific skills. Players can click on the icon to assign the command.

However, some unit may have skill that requires specifying a target on the main screen. After clicking on the command icon, the player left click on the main screen for targeting, or right click to cancel the skill.

Figure 3: Some units` skills may require specifying target zone

The following gestures are used on the command board:

1. Tapping on icons are treated as left clicking.
2. Pressing down at the command board, dragging to the main screen, then releasing. This will trigger a left click on the pressing down position, and another left click on the releasing position as specifying the target. If the gesture is released inside the command board, the skill will be cancelled.

2.2.4 Unit board

If a bunch of units is selected, each unit will be displayed as an icon in the unit board. Left clicking on the icon selects the corresponding unit, and double left clicking selects all the unit in the same type. Only 24 unit can be displayed at the same time. If the limit is exceeded, new pages will be created, and players can view different pages by clicking on the page markers.

Figure 4: Page markers in the unit board

If more than one type of unit is selected, units are arranged in subgroup and displayed hierarchically. Only the highlighted units will be served by the command board. Players roll back and forth among different unit types with the hotkey ***Tab*** and ***Shift + Tab***.

These commands are redesigned as the following:

1. Single tapping as left to select a single unit from the board and switch to other pages.
2. Double tapping to select a subgroup.
3. Sliding left to replace ***Tab*** and sliding right to replace ***Shift + Tab***

2.2.5 Control group

Control group is for quick access to units. By assigning unit into control groups, players can select these units by hitting the number key or clicking on the group icon, and double hit or click will jump the camera to where the units are on the map. Selected units can be assigned to a new group by hotkey ***Ctrl + number key*** or added to an existing control group by ***Shift + number key***. Maximum ten control groups can be created.

The gesture commands for control groups are:

1. Long press on a group to create the group.
2. Single tapping to select a group and double tapping to double click on the group.

2.3 Implementation

2.3.1 ApowerMirror

ApowerMirror is a screen mirroring app which is capable to mirror PC screen to the smartphone. The entire app is implanted as an activity of the project to fulfill the video stream feature. Technically, I used ***PackageManager***, a tool provided by Android studio to decoding the activity information of ApowerMirror, then launched it by issuing an ***intent***.

```
private void openApp(String packageName) throws PackageManager.NameNotFoundException {
    PackageManager pm = getPackageManager();
    Intent intent = pm.getLaunchIntentForPackage(packageName);
    startActivity(intent);
}
```

Figure 5: Code for launching an external app

2.3.2 Floating window: WindowManager and GestureDetector:

This project uses floating windows to overwrite control of ApowerMirror. By creating a transparent floating window on the top of ApowerMirror, gestures will be captured by the window instead of the screen mirroring app, so that I could handle the gesture recognition on the floating window instead of in the ApowerMirror, while player can still see the PC screen mirrored on ApowerMirror.

The floating window could be implemented by the WindowManager, an API to interact with Android window management system[3]. It takes any ***View*** object as input as well as a set of parameters to specify the position and size of the floating window. In this case, I programmed 5 self-defined ***View*** objects, representing respective sections of the gaming interface and displayed them as transparent floating windows.

```
windowManager = (WindowManager) this.getSystemService(Context.WINDOW_SERVICE);
//main screen
gestureController = new GestureController( context: this);
//miniMap
MiniMap = new miniMap( context: this);
//unit board
UnitBoard = new unitBoard( context: this);
//command board
CommandBoard = new commandBoard( context: this);
//control group
ControlGroup = new controlGroup[10];
for(int i = 0; i<10; i++){
    ControlGroup[i] = new controlGroup( context: this,  id: (i+1)%10);
}

//add minimap
WindowManager.LayoutParams windowParameters_MiniMap = new WindowManager.LayoutParams();

windowParameters_MiniMap.type = WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY;

windowParameters_MiniMap.format = PixelFormat.TRANSLUCENT;
windowParameters_MiniMap.width = 270;
windowParameters_MiniMap.height = 255;
windowParameters_MiniMap.x = -945;
windowParameters_MiniMap.y = 400;
windowParameters_MiniMap.flags |= WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
        | WindowManager.LayoutParams.FLAG_NOT_TOUCH_MODAL;

windowManager.addView(MiniMap,windowParameters_MiniMap);
```

Figure 6: Code for creating floating window

Everyone of windows above possess a unique set of gesture recognition rule. A GestureDetector is attached to each of them to monitoring touch events. It provides callback functions for gestures such as single tap, double tap, long press, scroll, etc. When a gesture is detected, I can send a message to the PC in the callbacks to announce it what in game command should be performed.

```java
// 5. 用户按下触摸屏 & 拖动
public boolean onScroll(MotionEvent e1, MotionEvent e2,
                        float distanceX, float distanceY) {
    Log.i( tag: "MyGesture1",  msg: "onScroll:");
    if(scroll_disabled){
        return true;
    }
    if(!pointer_down){
        int x = Math.round(e2.getX()-e1.getX());
        int y = Math.round(e2.getY()-e1.getY());
        if(Math.abs(x)>2*Math.abs(y)){
            y = 0;
        }
        if(2*Math.abs(x)<Math.abs(y)){
            x = 0;
        }
        MySocket.subhandler.obtainMessage( what: 1, obj: "14,"+ x +","+ y).sendToTarget();
    }
    else{
        int pointerCount = e2.getPointerCount();
        for (int i = 0; i < pointerCount; i++) {
            int pointerId = e2.getPointerId(i);
            if (pointerId == mActivePointerId) {
                int x = Math.round(e2.getX(i));
                int y = Math.round(e2.getY(i));
                MySocket.subhandler.obtainMessage( what: 1, obj: "15,"+ x +","+ y).sendToTarget();
            }
        }
    }
```

Figure 7: Clip of the gesture detector

GestureDetector is also capable to record the pixel coordinate of touch events. With the resolution rate is given for both of the PC and smartphone, each pixel on phone could be mapped to an equivalent position on the PC, so that the touch event location can help me to further specify where an action should be performed on the PC.

2.3.3 Socket

This project use socket for communication between the smartphone and the PC. As it is a mandatory of Android system, the socket is implemented in a separated thread. To communicate between the main thread and the socket thread, I used ***Handler*** to transmit data to the socket. In the GestureDetector mentioned above, the information of touch event is sent to the message pool first, then the socket obtains it from the pool and lastly the message is sent to the PC.

```
public static Handler subhandler;
5 usages
public static boolean lock = true;

private GestureDetector mGestureDetector;

1 usage
public MySocket() {

    //initalize handler

    //create a socket thread
    (Thread) run() → {
            Looper.prepare();

            try {
                acceptServer();
            } catch (IOException e) {
                e.printStackTrace();
            }

            subhandler = (Handler) handleMessage(msg) → {
                    if(lock){
                        lock = !lock;
                        try{
                            DataTransfer(msg.obj.toString());
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        try {
                            Thread.sleep( millis: 80);
```

Figure 8: Clip of socket

Also, I observed that in some gestures such as scrolling, a sequence of motion events will be triggered rapidly so that messages were sent at a high frequency. In this case, the messages close to each other may overlap at the receive end. To avoid that, I set a short delay to ensure

messages are sent only when the previous one is completely received. Messages intends to be sent in between the delay will be dropped to ensure the real-time.

2.3.4 Windows API

As StarCraft II does not provide any programmatic interface for players to access the game status and using external tool to access illegally will cause banning on players` account, the PC end program must reach the game from operating system level. Thus, commands in game will be performed by calling Windows APIs of keyboard and mouse. In this project, I used ***pynput.***

3. Evaluation

3.1 Value of this project

This project successfully explored a technical approach to take control over any PC game with smartphones. It requires four components to establish such a system: a screen mirroring app for synchronizing the screen between PCs and smartphones, a socket for communication, floating windows to overwrite input handle of the smartphone and APIs to imitate PC input. It is technically simple while does not need the game to provide any official interface for third-party programs. Theoretically, this technique is not necessarily limited to games. It is applicable for controlling any PC program with smartphones.

3.2 Potential improvements

Honestly, this project still has a lot of room for improvement. One of the unsettled problems is, the screen mirroring and socket transmission seems to have a delay around half a second, which unneglectable affect the real-time of the gameplay. I have identified that this may be caused by the interference between channels of screen mirroring and the socket. It was beyond my current capability to dig out more details behind that. Also, as the time is constrained, I introduced an entire screen mirror app to the project instead of implementing the mirroring feature on myself. It would be desirable for me to learn and integrate the screen mirroring feature to one app in the future.

4. Learning outcomes and trivia

In this project, most of my work is related to low-level layers of Android system, for example, the floating window, motion events and intents to calling one program in another.

To fully understand these mechanisms, I did a lot of experiments to study on them, such as how the coordinate of a motion event is generated, whether a floating window can be created on the top of another or not, etc. This not only trained my programming skills, but also led me to a more insightful view of the Android system.

At the time I came up with the idea of this project, I realized that I needed to examine the viability of it. After searching through forums such Stack Overflow, Github or CSDN, it seems no one raised up such an idea before. Honestly, I was quite doubtful at that time that if I can eventually accomplish the idea, but luckily, it turned out I made it with my effort. Although my project does not have a delicate layout or involve some fantastic algorithms, exploring and finally realizing an original idea is equally meaningful and fulfilling for me.

5. Reference

[1] "Real-time strategy," Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_strategy. [Accessed: Apr. 20, 2023].

[2] "Hotkeys per Race," Liquipedia, Jan. 31, 2023. [Online]. Available: https://liquipedia.net/starcraft2/Hotkeys_per_Race. [Accessed: Apr. 20, 2023].

[3] "WindowManager," Android Developer, 2023. [Online]. Available: https://developer.android.com/reference/android/view/WindowManager. [Accessed: Apr. 20, 2023].

6. Appendix: Market survey design:

Q1. How many years did you play StarCraftII?

A. Less than 1 year B. 1-2 years C. 3-4 years D. more than 4 years

Q2. What is your average APM on ladder?

A. <100 B. 100-120 C. 120-150 D. 150-180 E. 180-200 F. >200

Q3. What is your highest APM ever been recorded in a ladder game?

A. <150 B. 150-180 C. 180-200 D. 200-250 E. 250-300 F. >300

Q4. Did you struggle with your micro techniques?

A. never B. rarely C. sometimes D. often E.

always

Q5. Please mark the difficulty for each of the following micro tasks (1 for easy and 5 for unmanageable)

a) Use multiple control groups.
b) Kiting
c) Control caster unit, i.e. Sentry, Raven, Viper
d) Focusing fire on high threat enemy unit.
e) Splitting units to avoid area of effect (AOE) damage.
f) Multitasking
g) Flanking and surrounding