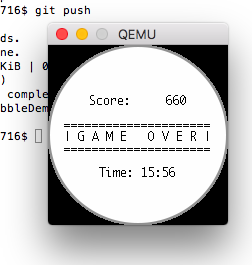
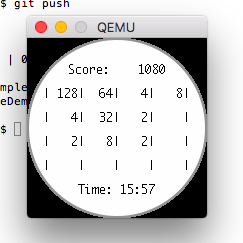
可穿戴计算实验报告

班级：计算机1403

学号：41455077

姓名：陈正

## 在实验任务之前，我希望先给老师看一个有意思的东西：



### 如您所见，这是我制作在pebble上的一个2048小游戏app。

怎么说呢，比起找着实验任务与给好的半成品代码一步步来，我个人更喜欢参考这官方api跟着兴趣做点东西。这个app实现了诸如logo图片的呈现、动画效果、layer层的切换、内容的动态变更、时间的显示、按键监听、外部字体的引用等功能，并且是在自己的mac上搭建了离线的开发环境，希望能够作为本次实验的收获与成果总结。

代码开源托管在了GitHub上：<https://github.com/zhengxiaoyao0716/PebbleDemo>

下面请允许我对app源码做简单的说明介绍

### 首先是包的公共头文件，定义了几个全局变量：

#ifndef \_\_package\_H

#define \_\_package\_H

#include <pebble.h>

extern Window\* mainWindow;

extern GRect bounds;

extern GBitmap\* bitmap;

extern BitmapLayer\* bitmapLayer;

extern TextLayer\* textLayer;

extern GFont font;

#endif

### 接着是app入口：main.c

这个模块里主要负责管理app生命周期以及app基本事件的注册，不涉及具体的游戏逻辑。

#include "game/package.h"

#include "game/button.h"

#include "game/frame.h"

#include "game/game.h"

Window\* mainWindow;

GRect bounds;

GBitmap \*bitmap;

BitmapLayer\* bitmapLayer;

static void clickConfigProvider(void \*context) {

window\_single\_click\_subscribe(BUTTON\_ID\_UP, onClick.up);

window\_single\_click\_subscribe(BUTTON\_ID\_DOWN, onClick.down);

window\_single\_click\_subscribe(BUTTON\_ID\_BACK, onClick.back);

window\_single\_click\_subscribe(BUTTON\_ID\_SELECT, onClick.select);

}

static void onWindowLoad(Window \*window) {

// 配置按键

window\_set\_click\_config\_provider(window, clickConfigProvider);

// 窗口信息

Layer \*layer = window\_get\_root\_layer(window);

bounds = layer\_get\_bounds(layer);

// bitmap图层

GRect fromRect = GRect(0, 0, bounds.size.w, bounds.size.h);

GRect toRect = GRect(0, bounds.size.h, bounds.size.w, bounds.size.h);

bitmapLayer = bitmap\_layer\_create(fromRect);

// bitmap对象

bitmap = gbitmap\_create\_with\_resource(RESOURCE\_ID\_MY\_LOGO);

bitmap\_layer\_set\_compositing\_mode(bitmapLayer, GCompOpSet);

bitmap\_layer\_set\_bitmap(bitmapLayer, bitmap);

// logo动画

PropertyAnimation \*propAnim = property\_animation\_create\_layer\_frame((Layer \*)bitmapLayer, &fromRect, &toRect);

Animation \*anim = property\_animation\_get\_animation(propAnim);

const int delay\_ms = 1000;

const int duration\_ms = 3000;

animation\_set\_curve(anim, AnimationCurveEaseOut);

animation\_set\_delay(anim, delay\_ms);

animation\_set\_duration(anim, duration\_ms);

animation\_schedule(anim);

// 添加图层

layer\_add\_child(window\_get\_root\_layer(window), bitmap\_layer\_get\_layer(bitmapLayer));

}

static void onWindowUnload(Window \*window) {

if (bitmap != NULL) gbitmap\_destroy(bitmap);

if (bitmapLayer != NULL) bitmap\_layer\_destroy(bitmapLayer);

gameLoop.finish();

}

static void init() {

// 注册时间服务

tick\_timer\_service\_subscribe(SECOND\_UNIT, frame.refresh);

// 创建窗口

mainWindow = window\_create();

// 注册窗口事件

window\_set\_window\_handlers(mainWindow, (WindowHandlers) {

.load = onWindowLoad,

.unload = onWindowUnload

});

// 显示窗口

window\_stack\_push(mainWindow, true);

}

static void deinit() {

// 销毁窗口

window\_destroy(mainWindow);

}

int main(void) {

init();

app\_event\_loop();

deinit();

}

### frame模块则负责动画帧的渲染内容的调度

frame.h

#ifndef \_\_FRAME\_H

#define \_\_FRAME\_H

#include "package.h"

typedef struct {

void (\* refresh)(struct tm \*tick\_time, TimeUnits units\_changed);

} Frame;

extern Frame frame;

#endif

frame.c

#include "frame.h"

#include "game.h"

static int timeAxis = -3;

static void refresh(struct tm \*tick\_time, TimeUnits units\_changed) {

timeAxis++;

if (timeAxis > 0) {

gameLoop.update(timeAxis);

} else if (timeAxis == 0) {

gameLoop.start();

gbitmap\_destroy(bitmap);

layer\_remove\_from\_parent((Layer \*)bitmapLayer);

bitmap\_layer\_destroy(bitmapLayer);

bitmap = NULL;

bitmapLayer = NULL;

}

}

Frame frame = {

refresh

};

### game模块负责游戏主循环的调度以及游戏资源的管理

game.h

#ifndef \_\_GAME\_H

#define \_\_GAME\_H

#include "package.h"

typedef struct {

void (\* start)();

void (\* update)(int timeAxis);

void (\* finish)();

} GameLoop;

extern GameLoop gameLoop;

#endif

game.c

#include "game.h"

#include "board.h"

bool running = false;

TextLayer\* textLayer;

GFont font;

static void start() {

running = true;

GRect gRect = GRect(0, 0, bounds.size.w, bounds.size.h);

textLayer = text\_layer\_create(gRect);

layer\_add\_child(window\_get\_root\_layer(mainWindow), text\_layer\_get\_layer(textLayer));

font = fonts\_load\_custom\_font(resource\_get\_handle(RESOURCE\_ID\_MONACO\_12));

text\_layer\_set\_font(textLayer, font);

text\_layer\_set\_text\_alignment(textLayer, GTextAlignmentCenter);

board.begin();

}

static void update(int timeAxis) {

time\_t temp = time(NULL);

struct tm \*tickTime = localtime(&temp);

strftime(board.timeStr, sizeof(board.timeStr), clock\_is\_24h\_style() ? "%H:%M" : "%I:%M", tickTime);

strcpy(board.content, board.boardStr);

strcat(board.content, board.timeStr);

text\_layer\_set\_text(textLayer, board.content);

}

static void finish() {

running = false;

text\_layer\_destroy(textLayer);

textLayer = NULL;

}

// static bool isRunning() {return running;}

GameLoop gameLoop = {

start, update, finish

};

### board模块负责棋盘的刻画与变更，也就是2048游戏的主逻辑所在

board.h

#ifndef \_\_BOARD\_H

#define \_\_BOARD\_H

#include "package.h"

typedef struct {

char content[128];

char timeStr[8];

char boardStr[120];

void (\* begin)();

void (\* trigger)(char action);

} Board;

extern Board board;

#endif

board.c

#include "board.h"

int score = 0;

int content[4][4] = {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};

static void writeScore(int scoreIndex) {

int scoreIter = score;

while (scoreIter != 0) {

board.boardStr[scoreIndex--] = scoreIter % 10 + 48;

scoreIter = scoreIter / 10;

}

}

static void putContent()

{

writeScore(15);

int firstNumIndex = 22;

int x, y;

for (x = 0; x<= 3; x++)

{

int lineFirstIndex = firstNumIndex + 23 \* x;

for (y = 0; y<= 3; y++)

{

int numIndex = lineFirstIndex + 5 \* y;

int gridLeftIndex = numIndex - 4;

int num = content[x][y];

while (num != 0) {

board.boardStr[numIndex--] = num % 10 + 48;

num = num / 10;

}

while (numIndex > gridLeftIndex) {

board.boardStr[numIndex--] = ' ';

}

}

}

strcpy(board.content, board.boardStr);

strcat(board.content, board.timeStr);

text\_layer\_set\_text(textLayer, board.content);

}

static void change(char action)

{

int x, y;

switch (action)

{

case 'w':

case '2': break;

case 'a':

case '4':

{

for (x = 0; x<= 2; x++)

{

for (y = x+1; y<= 3; y++)

{

content[x][y] += content[y][x];

content[y][x] = content[x][y]-content[y][x];

content[x][y] -= content[y][x];

}

}

break;

}

case 'd':

case '6':

{

for (x = 0; x<= 2; x++)

{

for (y = 0; y<= 2-x; y++)

{

content[x][y] += content[3-y][3-x];

content[3-y][3-x] = content[x][y]-content[3-y][3-x];

content[x][y] -= content[3-y][3-x];

}

}

break;

}

case 's':

case '8':

{

for (x = 0; x<= 1; x++)

{

for (y = 0; y<= 3; y++)

{

content[x][y] += content[3-x][y];

content[3-x][y] = content[x][y]-content[3-x][y];

content[x][y] -= content[3-x][y];

}

}

break;

}

// default: exit(0);

}

}

static int merge()

{

int flag = 0;

int x, y;

for (y = 0; y<= 3; y++)

{

int lastX = -1, i = 0;

int temp[5] = {0};

for (x = 0; x <= 3; x++)

{

if (content[x][y])

{

temp[i] = content[x][y];

content[x][y] = 0;

i++;

lastX = x;

}

}

if (i - 1 != lastX) flag++;

x = 0;

for (i = 0; i<= 3; i++)

{

if (temp[i])

{

if (temp[i] == temp[i+1])

{

temp[i] += temp[i+1];

score += temp[i];

temp[i+1] = 0;

flag++;

}

content[x][y] = temp[i];

x++;

}

}

}

return flag;

}

static int giveNew()

{

int x, y, i = 0, \*temp[16];

for (x = 0; x<= 3; x++ )

{

for (y = 0; y<= 3; y++)

{

if (!content[x][y])

{

temp[i] = &content[x][y];

i++;

}

}

}

if (i != 0) {

i = rand()%i;

\*temp[i] = (rand()%2+1)\*2;

return 1;

} else return 0;

}

static int isOver()

{

int x;

for (x = 0; x<= 3; x++)

{

if ((content[x][0] == content[x][1])||(content[x][1] == content[x][2])||(content[x][2] == content[x][3]))

{

return 0;

}

}

return 1;

}

static void begin()

{

srand(time(NULL));

giveNew();

putContent();

}

static void trigger(char action) {

change(action);

if ((!merge() || !giveNew()) && isOver()) {

strcpy(board.boardStr, "\n\n\n\nScore: 0\n\n=====================\n| G A M E O V E R |\n=====================\n\nTime: ");

writeScore(17);

} else {

change(action);

putContent();

}

}

Board board = {

"", "",

"\n\nScore: 0\n\n| | | | |\n\n| | | | |\n\n| | | | |\n\n| | | | |\n\nTime: ",

begin, trigger

};

### button模块，顾名思义，负责按键事件的注册与监听

button.h

#ifndef \_\_BUTTON\_H

#define \_\_BUTTON\_H

#include "package.h"

typedef struct {

void (\* up)(ClickRecognizerRef recognizer, void \*context);

void (\* down)(ClickRecognizerRef recognizer, void \*context);

void (\* back)(ClickRecognizerRef recognizer, void \*context);

void (\* select)(ClickRecognizerRef recognizer, void \*context);

} OnClick;

extern OnClick onClick;

#endif

button.c

#include "button.h"

#include "board.h"

static void onUpClick(ClickRecognizerRef recognizer, void \*context) {

board.trigger('2');

}

static void onDownClick(ClickRecognizerRef recognizer, void \*context) {

board.trigger('8');

}

static void onBackClick(ClickRecognizerRef recognizer, void \*context) {

board.trigger('4');

}

static void onSelectClick(ClickRecognizerRef recognizer, void \*context) {

board.trigger('6');

}

OnClick onClick = {

onUpClick, onDownClick, onBackClick, onSelectClick

};

### 最后，package.json里申明了一些资源配置，主要是resources下的一个等宽字体和一个logo图片：

"resources": {

"media": [

{

"type": "bitmap",

"name": "MY\_LOGO",

"file": "image/logo.png",

"memoryFormat": "Smallest",

"spaceOptimization": "memory"

},

{

"type": "font",

"name": "MONACO\_12",

"file": "font/MONACO.TTF"

}

]

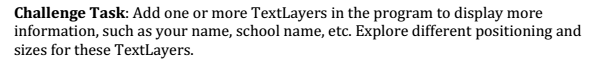
}

## 实验感悟：

这次的实验总得来说也没什么难点，官方api还算详细，理解起来也比较容易，再加上官网上给出的一些优秀案例源码，很容易就可以上手。并且基于智能手表做小游戏app也挺好玩的。

后面这部分说实话是参照了同学的代码，因为实在没什么意思，就是照着任务要求一步步做而已。我觉得没必要再仔细介绍了

TASK1



#include <pebble.h>

Window \*window;

TextLayer \*text\_layer;

void init() {

window = window\_create();

text\_layer = text\_layer\_create(GRect(0, 0, 144, 40));

text\_layer\_set\_text(text\_layer, "USTB\nCS1403\_YangJinGu");

layer\_add\_child(window\_get\_root\_layer(window), text\_layer\_get\_layer(text\_layer));

window\_stack\_push(window, true);

}

void deinit() {

text\_layer\_destroy(text\_layer);

window\_destroy(window);

}

int main() {

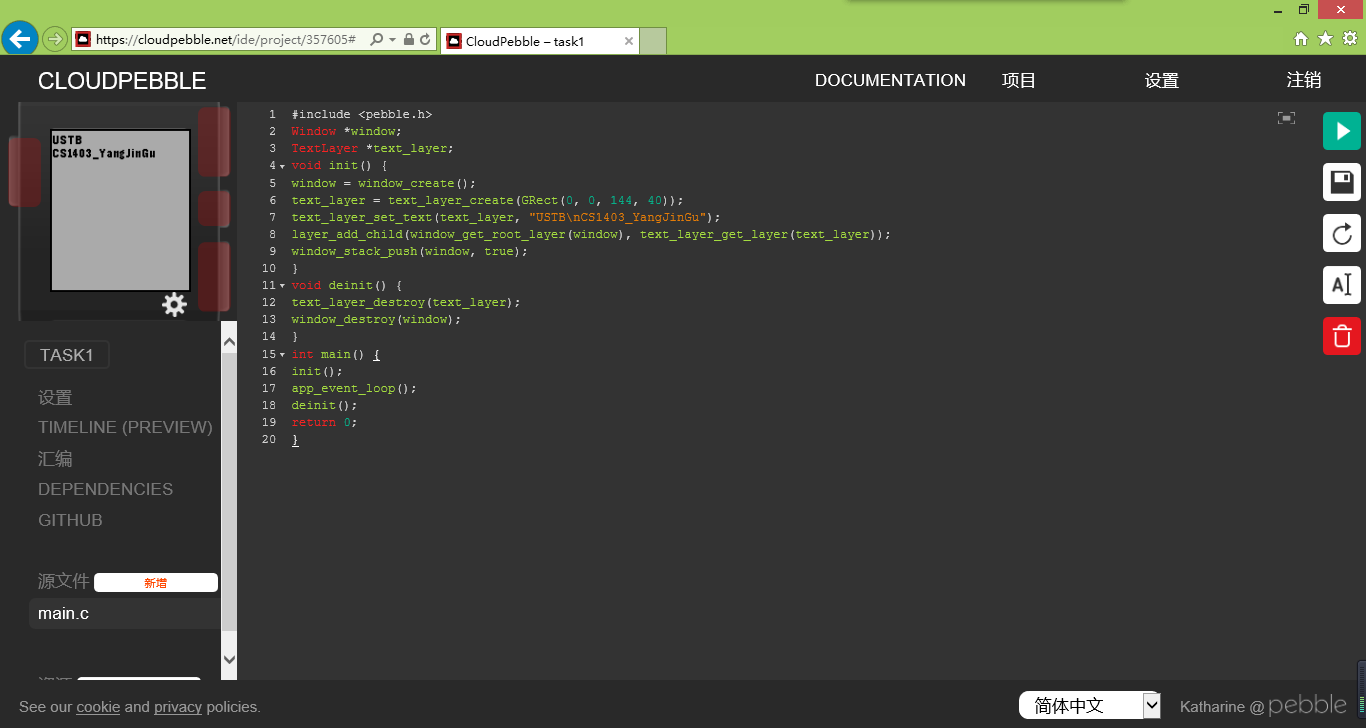
init();

app\_event\_loop();

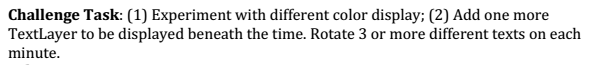
deinit();

return 0;

}



TASK2



#include <pebble.h>

Window \*window;

// Global variables

char buffer[] = "00:00";

TextLayer \*text\_layer1;

// update the time display when the minute ticks is over

// The ‘tick\_time’ argument contains the time data and

// ‘units\_changed’ contains which unit was changed

void tick\_handler(struct tm \*tick\_time, TimeUnits units\_changed)

{

//Format the buffer string using tick\_time as the time source

strftime(buffer, sizeof("00:00"), "%H:%M", tick\_time);

//Change the TextLayer text to show!

int seconds = tick\_time->tm\_sec;

if(seconds > 0 && seconds < 10)

{

text\_layer\_set\_text(text\_layer1, "USTB");

}

else if(seconds > 40 && seconds < 50)

{

text\_layer\_set\_text(text\_layer1, "SC1403\_41455069");

}

else

{

//Change the TextLayer text to show the new time!

text\_layer\_set\_text(text\_layer1, buffer);

}

}

void window\_load(Window \*window)

{

// creation of the Window's elements

// Make the watch face pretty

text\_layer1 = text\_layer\_create(GRect(0, 53, 132, 168));

text\_layer\_set\_background\_color(text\_layer1, GColorClear);

text\_layer\_set\_text\_color(text\_layer1, GColorRed);

text\_layer\_set\_text\_alignment(text\_layer1, GTextAlignmentCenter);

text\_layer\_set\_font(text\_layer1,

fonts\_get\_system\_font(FONT\_KEY\_BITHAM\_42\_BOLD));

layer\_add\_child(window\_get\_root\_layer(window), (Layer\*) text\_layer1);

}

void window\_unload(Window \*window)

{

// destroy the Window's elements

text\_layer\_destroy(text\_layer1);

}

void init()

{

//Initialize the app elements here!

window = window\_create();

window\_set\_window\_handlers(window, (WindowHandlers) {

.load = window\_load,

.unload = window\_unload,

}

);

window\_stack\_push(window, true);

tick\_timer\_service\_subscribe(MINUTE\_UNIT,(TickHandler)tick\_handler);

}

void deinit()

{

window\_destroy(window);

tick\_timer\_service\_unsubscribe();

}

int main(void)

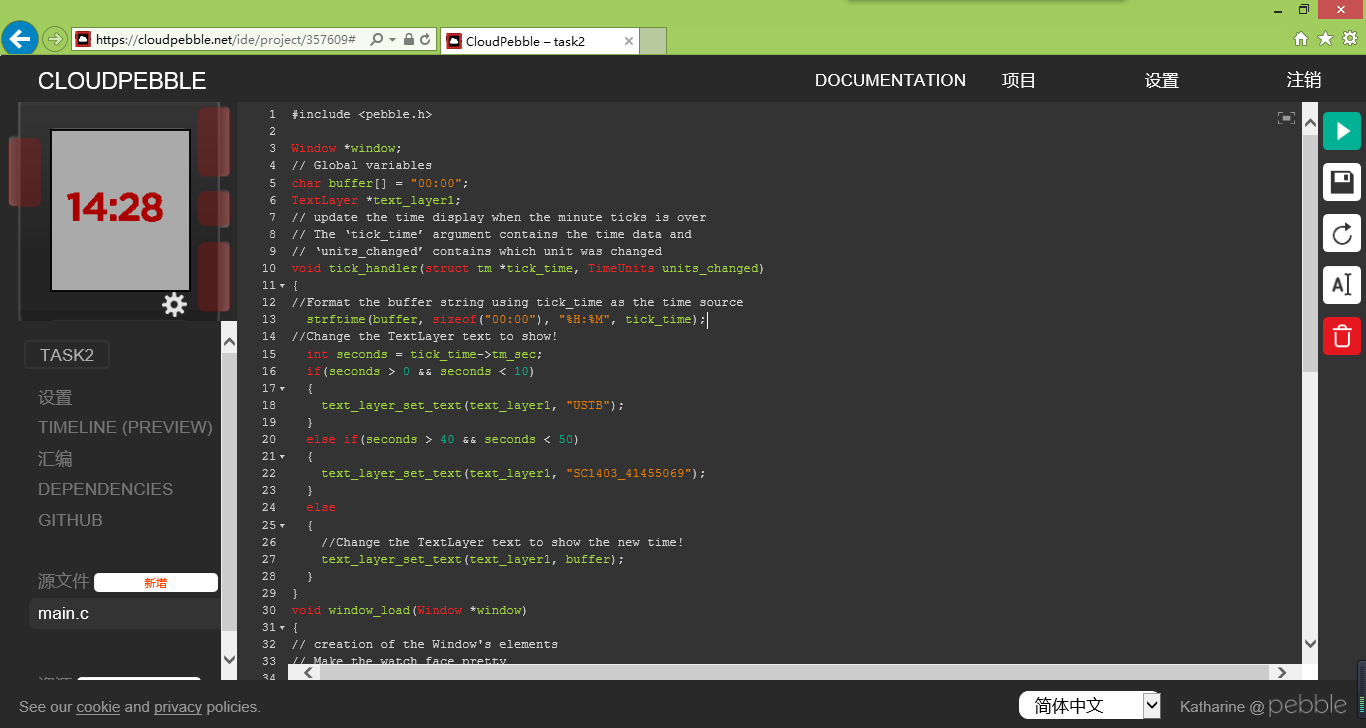
{

init();

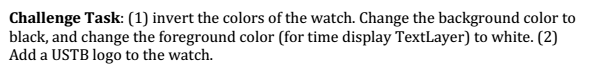
app\_event\_loop();

deinit();

}



TASK3



#include <pebble.h>

Window \*window;

// Global variables

char buffer[] = "00:00";

TextLayer \*text\_layer;

GBitmap \*ustb\_bitmap;

BitmapLayer \*ustb\_layer;

// update the time display when the minute ticks is over

// The ‘tick\_time’ argument contains the time data and

// ‘units\_changed’ contains which unit was changed

void tick\_handler(struct tm \*tick\_time, TimeUnits units\_changed)

{

//Format the buffer string using tick\_time as the time source

strftime(buffer, sizeof("00:00"), "%H:%M", tick\_time);

//Change the TextLayer text to show the new time!

text\_layer\_set\_text(text\_layer, buffer);

}

void window\_load(Window \*window)

{

// Make sure this block comes last

text\_layer = text\_layer\_create(GRect(0, 80, 144, 168));

text\_layer\_set\_background\_color(text\_layer, GColorBlack);

text\_layer\_set\_text\_color(text\_layer, GColorWhite);

text\_layer\_set\_text\_alignment(text\_layer, GTextAlignmentCenter);

text\_layer\_set\_font(text\_layer,

fonts\_get\_system\_font(FONT\_KEY\_BITHAM\_42\_BOLD));

layer\_add\_child(window\_get\_root\_layer(window), (Layer\*) text\_layer);

ustb\_bitmap = gbitmap\_create\_with\_resource(RESOURCE\_ID\_USTB);

// Create BitmapLayers to show GBitmaps and add to Window

// Sample images are 80 x 80 pixels

ustb\_layer = bitmap\_layer\_create(GRect(32, 0, 80, 80));

bitmap\_layer\_set\_bitmap(ustb\_layer, ustb\_bitmap);

layer\_add\_child(window\_get\_root\_layer(window),

bitmap\_layer\_get\_layer(ustb\_layer));

}

void window\_unload(Window \*window)

{

// destroy the Window's elements

text\_layer\_destroy(text\_layer);

}

void init()

{

//Initialize the app elements here!

window = window\_create();

window\_set\_window\_handlers(window, (WindowHandlers) {

.load = window\_load,

.unload = window\_unload,

});

tick\_timer\_service\_subscribe(MINUTE\_UNIT,(TickHandler)tick\_handler);

window\_stack\_push(window, true);

}

void deinit()

{

//Destroy GBitmaps

gbitmap\_destroy(ustb\_bitmap);

//Destroy BitmapLayers

bitmap\_layer\_destroy(ustb\_layer);

window\_destroy(window);

tick\_timer\_service\_unsubscribe();

}

int main(void)

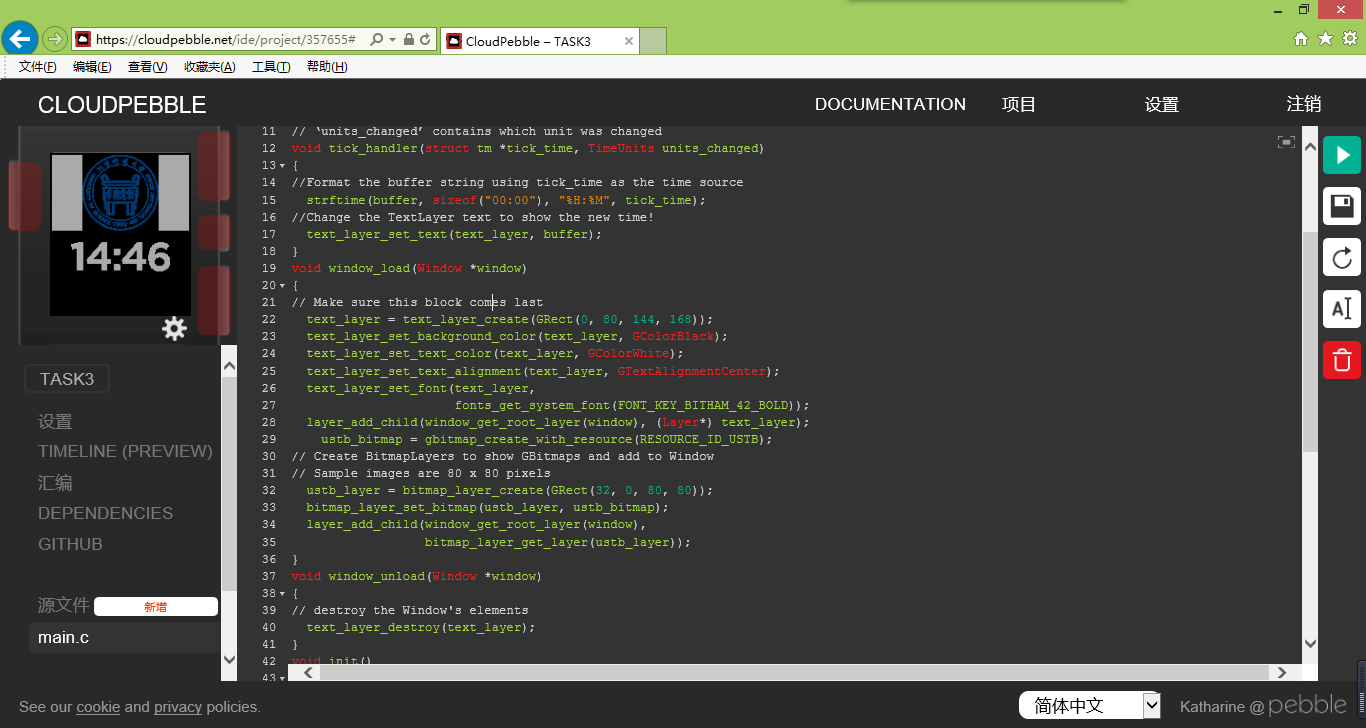
{

init();

app\_event\_loop();

deinit();

}



TASK4



#include <pebble.h>

#include <pebble.h>

Window \*window;

char buffer[] = "00:00";

TextLayer \*text\_layer;

//InverterLayer \*inv\_layer;

void on\_animation\_stopped(Animation \*anim, bool finished, void \*context)

{

//Free the memory used by the Animation

property\_animation\_destroy((PropertyAnimation\*) anim);

}

void animate\_layer(Layer \*layer, GRect \*start, GRect \*finish, int duration, int delay)

{

//Declare animation

PropertyAnimation \*anim = property\_animation\_create\_layer\_frame(layer, start, finish);

//Set characteristics

animation\_set\_duration((Animation\*) anim, duration);

animation\_set\_delay((Animation\*) anim, delay);

//Set stopped handler to free memory

AnimationHandlers handlers = {

//The reference to the stopped handler is the only one in the array

.stopped = (AnimationStoppedHandler) on\_animation\_stopped

};

animation\_set\_handlers((Animation\*) anim, handlers, NULL);

//Start animation!

animation\_schedule((Animation\*) anim);

}

void tick\_handler(struct tm \*tick\_time, TimeUnits units\_changed)

{

//Format the buffer string using tick\_time as the time source

strftime(buffer, sizeof("00:00"), "%H:%M", tick\_time);

int seconds = tick\_time->tm\_sec;

if(seconds == 59)

{

//Slide offscreen to the right

GRect start = GRect(288, 53, 144, 168);

GRect finish = GRect(144, 53, 144, 168);

animate\_layer(text\_layer\_get\_layer(text\_layer), &start, &finish, 300, 500);

}

else if(seconds == 0)

{

//Change the TextLayer text to show the new time!

text\_layer\_set\_text(text\_layer, buffer);

//Slide onscreen from the left

GRect start = GRect(144, 53, 144, 168);

GRect finish = GRect(0, 53, 144, 168);

animate\_layer(text\_layer\_get\_layer(text\_layer), &start, &finish, 300, 500);

}

else

{

//Change the TextLayer text to show the new time!

text\_layer\_set\_text(text\_layer, buffer);

}

}

void window\_load(Window \*window)

{

text\_layer = text\_layer\_create(GRect(0, 53, 132, 168));

text\_layer\_set\_background\_color(text\_layer, GColorClear);

text\_layer\_set\_text\_color(text\_layer, GColorBlue);

text\_layer\_set\_text\_alignment(text\_layer, GTextAlignmentCenter);

text\_layer\_set\_font(text\_layer,

fonts\_get\_system\_font(FONT\_KEY\_BITHAM\_42\_BOLD));

layer\_add\_child(window\_get\_root\_layer(window),(Layer\*)text\_layer);

}

void window\_unload(Window \*window)

{

text\_layer\_destroy(text\_layer);

}

void init()

{

//Initialize the app elements here!

window = window\_create();

window\_set\_window\_handlers(window, (WindowHandlers) {

.load = window\_load,

.unload = window\_unload,

});

tick\_timer\_service\_subscribe(SECOND\_UNIT, (TickHandler) tick\_handler);

window\_stack\_push(window, true);

}

void deinit()

{

//De-initialize elements here to save memory!

tick\_timer\_service\_unsubscribe();

}

int main(void)

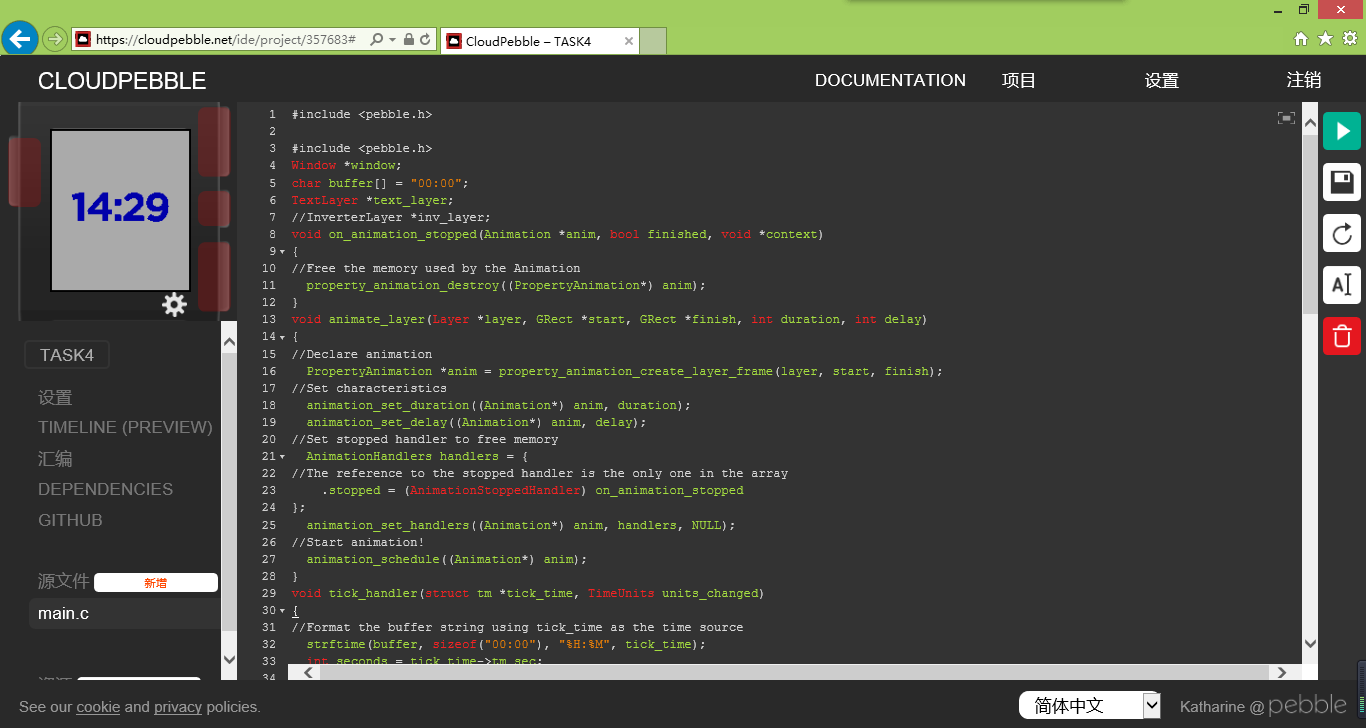
{

init();

app\_event\_loop();

deinit();

}



TASK5



#include <pebble.h>

Window \*window;

char buffer[] = "00:00";

TextLayer \*text\_layer;

TextLayer \*text\_layer, \*square\_layer;

AppTimer \*timer;

const int square\_size = 10;

const int delta = 40;

int dx = 1;

int dy = 0;

void timer\_callback(void \*data) {

// Get current position

GRect current = layer\_get\_frame(text\_layer\_get\_layer(square\_layer));

//Check to see if we have hit the edges

if(current.origin.x > 144 - square\_size )

{

dx = 0;

dy = 1;

if(current.origin.y > 144 - square\_size)

{

dx = -1;

dy = 0;

}

// //Reverse

}

else if(current.origin.x < 0)

{

dx = 0; //Forwards

dy = -1;

if(current.origin.y < 0)

{

dx = 1;

dy = 0;

}

}

else if(current.origin.y > 144 - square\_size)

{

dx = -1;

dy = 0;

if(current.origin.x < 0)

{

dx = 0; //Forwards

dy = -1;

}

}

//Move the square to the next position, modifying the x value

GRect next = GRect(current.origin.x + dx, current.origin.y + dy, square\_size, square\_size);

layer\_set\_frame(text\_layer\_get\_layer(square\_layer), next);

//Register next execution

timer = app\_timer\_register(delta, (AppTimerCallback) timer\_callback, NULL);

}

void tick\_handler(struct tm \*tick\_time, TimeUnits units\_changed)

{

//Format the buffer string using tick\_time as the time source

strftime(buffer, sizeof("00:00"), "%H:%M", tick\_time);

//Change the TextLayer text to show the new time!

text\_layer\_set\_text(text\_layer, buffer);

}

void window\_load(Window \*window)

{

text\_layer = text\_layer\_create(GRect(0, 53, 132, 168));

text\_layer\_set\_background\_color(text\_layer, GColorClear);

text\_layer\_set\_text\_color(text\_layer, GColorBlue);

text\_layer\_set\_text\_alignment(text\_layer, GTextAlignmentCenter);

text\_layer\_set\_font(text\_layer,

fonts\_get\_system\_font(FONT\_KEY\_BITHAM\_42\_BOLD));

layer\_add\_child(window\_get\_root\_layer(window), (Layer\*) text\_layer);

//Create the square layer

square\_layer = text\_layer\_create(GRect(0, 0, square\_size, square\_size));

text\_layer\_set\_background\_color(square\_layer, GColorRed);

layer\_add\_child(window\_get\_root\_layer(window), text\_layer\_get\_layer(square\_layer));

//Start the square moving

timer = app\_timer\_register(delta, (AppTimerCallback) timer\_callback, NULL);

}

void window\_unload(Window \*window)

{

//We will safely destroy the Window's elements here!

text\_layer\_destroy(text\_layer);

//Cancel timer

app\_timer\_cancel(timer);

//Destroy square layer

text\_layer\_destroy(square\_layer);

}

void init()

{

//Initialize the app elements here!

window = window\_create();

window\_set\_window\_handlers(window, (WindowHandlers) {

.load = window\_load,

.unload = window\_unload,

});

tick\_timer\_service\_subscribe(MINUTE\_UNIT, (TickHandler) tick\_handler);

window\_stack\_push(window, true);

}

void deinit()

{

//De-initialize elements here to save memory!

tick\_timer\_service\_unsubscribe();

}

int main(void)

{

init();

app\_event\_loop();

deinit();

}

