class Effect extends egret.Sprite {

public constructor() {

super();

}

public effectNumber: number;

protected w: number;

protected h: number;

public frameCnt: number = 0;

public endCnt: number;

public doDraw: boolean = false;

// public matrix: egret.Matrix;

public color: number;

public drawBorderRect(x: number, y: number, width: number, height: number, thickness: number, color: number) {

this.graphics.clear();

this.graphics.beginFill(0x00000, 0);

this.graphics.lineStyle(thickness, this.color);

this.graphics.drawRect(0, 0, width, height);

this.graphics.endFill();

}

public update(): boolean {

return false;

}

}

class EffectLayerErase extends Effect {

private onComPlete: boolean = false;

public constructor(x: number, y: number, color: number) {

super();

this.endCnt = 25;

this.x = x;

this.y = y;

this.color = color;

this.drawBorderRect(0, 0, 120, 120, 10, this.color);

this.x = x + (this.width >> 1);

this.y = y + (this.height >> 1);

this.anchorOffsetX = this.width >> 1;

this.anchorOffsetY = this.height >> 1;

this.start();

}

public update(): boolean {

return this.onComPlete;

}

private start() {

// TweenLite.to(this, this.endCnt / 60, { scaleX: 6, scaleY: 6, onComplete: () => { this.onComPlete = true; } });

egret.Tween.get(this).to({ scaleX: 6, scaleY: 6, }, this.endCnt \* 10).call(() => { this.onComPlete = true })

}

}

class EffectManagerClass extends egret.Sprite {

public constructor() {

super();

this.vecEffect = [];

}

public vecEffect: Array<Effect>;

public update() {

var i = 0;

var ef: Effect = null;

var length = this.vecEffect.length;

i = 0;

while (i < length) {

ef = this.vecEffect[i];

if (ef.update()) {

console.log("removeEffect");

EffectManager.removeEffect(i);

i--;

length--;

}

i++;

}

}

public addEffect(effect: Effect) {

effect.effectNumber = this.vecEffect.length;

this.vecEffect.push(effect);

this.addChild(effect);

}

public removeEffect(effectNumber: number) {

var i = 0;

let effect = this.vecEffect.splice(effectNumber, 1);

if (effect) {

var length = this.vecEffect.length;

this.removeChild(effect[0]);

// i = effectNumber;

// while (i < length) {

// this.vecEffect[i].effectNumber--;

// i++;

// }

}

}

}

var EffectManager: EffectManagerClass = new EffectManagerClass();

class Input extends eui.Group {

public constructor() {

super();

// this.graphics.beginFill(0, 0);

// this.graphics.drawRect(0, 0, 480, 762);

this.percentWidth = 100;

this.percentHeight = 100;

this.addEventListener("touchBegin", this.mouseDownEvent, this);

this.addEventListener("touchEnd", this.mouseUpEvent, this);

this.addEventListener("touchMove", this.mouseMoveEvent, this);

this.addEventListener("touchReleaseOutside", this.mouseOutEvent, this);

// this.touchEnabled = true;

// this.touchThrough = true;

}

protected mouseDownEvent(param1: egret.TouchEvent): void {

}

protected mouseUpEvent(param1: egret.TouchEvent): void {

}

protected mouseOutEvent(param1: egret.TouchEvent): void {

}

protected mouseMoveEvent(e: egret.TouchEvent): void {

}

}

class InputManagerClass extends eui.Component {

public constructor() {

super();

this.percentWidth = 100;

this.percentHeight = 100;

// this.touchChildren = true;

// this.touchEnabled = false;

}

private input: Input;

public newInput(input: Input) {

this.removeChildren();

this.input = input;

if (input != null) {

this.addChild(input);

}

}

}

var InputManager: InputManagerClass = new InputManagerClass();

class Resource {

public static sounds: Array<egret.Sound>

public static ChipsSpriteSheet: egret.SpriteSheet;

public static Instance: Resource = new Resource();

public constructor() {

if (Resource.Instance) return Resource.Instance;

}

public static getBlockChips(x: number, y: number, w = 44, h = 44): eui.Image {

if (!this.ChipsSpriteSheet) this.ChipsSpriteSheet = new egret.SpriteSheet(RES.getRes("Chip\_png"))

let chip = this.ChipsSpriteSheet.getTexture(x + "\_" + y);

if (!chip) {

this.ChipsSpriteSheet.createTexture(x + "\_" + y, x, y, w, h);

chip = this.ChipsSpriteSheet.getTexture(x + "\_" + y);

};

return new eui.Image(chip);

}

public static getMusic(): Array<egret.Sound> {

if (!this.sounds) {

this.sounds = [RES.getRes("SoundC\_mp3"), RES.getRes("SoundD\_mp3"), RES.getRes("SoundE\_mp3"), RES.getRes("SoundF\_mp3"), RES.getRes("SoundG\_mp3"), RES.getRes("SoundA\_mp3"), RES.getRes("SoundB\_mp3"), RES.getRes("SoundCC\_mp3")];

this.sounds.forEach((sound) => {

sound.type = egret.Sound.EFFECT;

})

}

return this.sounds;

}

}

class Scene extends eui.Component {

public constructor() {

super();

this.percentHeight = 100;

this.percentWidth = 100;

}

public update() {

}

}

class SceneManagerClass extends eui.Component {

public constructor() {

super();

this.percentHeight = 100;

this.percentWidth = 100;

}

public scene: Scene;

private nextScene: Scene;

public newScene(next: Scene) {

this.nextScene = next;

if (this.scene != null) {

egret.Tween.get(this.scene).to({ alpha: 0 }, 300, egret.Ease.elasticOut).call(this.next, this);

}

else {

this.next();

}

}

public next() {

var sceneId = 0;

this.removeChildren();

this.scene = null;

this.scene = this.nextScene;

if (this.scene instanceof SetPuzzleScene) {

sceneId = 0;

}

else if (this.scene instanceof SetScoreScene) {

sceneId = 1;

}

else if (this.scene instanceof SetScore30Scene) {

SetScore30Scene.cnt = 30;

SetScore30Scene.isFinish = false;

sceneId = 2;

}

else if (this.scene instanceof SetScore1minScene) {

SetScore1minScene.cnt = -95;

SetScore1minScene.isFinish = false;

sceneId = 3;

}

else if (this.scene instanceof SetScore1comboScene) {

SetScore1comboScene.isFinish = false;

sceneId = 4;

}

Status.reset(sceneId);

this.nextScene.alpha = 0;

egret.Tween.get(this.nextScene).to({ alpha: 1 }, 200, egret.Ease.elasticOut);

this.addChild(this.scene);

}

}

var SceneManager: SceneManagerClass = new SceneManagerClass();

class SharedManagerClass {

public vecPuzzleClear: Array<number> = [];

public score: number;

public score1min: number;

public score30: number;

public score1combo: number;

public userName: string;

public sound: boolean = true;

public sendScoreTime: number = 0;

public sendScore30Time: number = 0;

public sendScore1minTime: number = 0;

public sendScore1comboTime: number = 0;

public dayScore: number;

public dayScore30: number;

public dayScore1min: number;

public dayScore1combo: number;

private static instance: SharedManagerClass;

public static getInstance() {

!this.instance && (this.instance = new SharedManagerClass())

return this.instance;

}

public constructor() {

}

public async init() {

try {

let i = 0;

while (i < 25) {

this.vecPuzzleClear.push(0);

i++;

}

// this.score = this.parseItem2Number("score");

// this.score1min = this.parseItem2Number("score30");

// this.score30 = this.parseItem2Number("score1min");

// this.score1combo = this.parseItem2Number("score1combo");

// this.getScore();

// this.getScore30();

// this.getScore1combo();

// this.getScore1min();

let resultData = await platform.getStorage("score");

this.score = resultData;

resultData = await platform.getStorage("score30")

this.score30 = resultData;

resultData = await platform.getStorage("score1min")

this.score1min = resultData;

resultData = await platform.getStorage("score1combo")

this.score1combo = resultData;

console.log("init:", this.score, this.score30, this.score1min, this.score1combo);

} catch (e) {

console.error(e);

}

}

public getPuzzleClearNum() {

return

}

private async getScore() {

let resultData = await platform.getStorage("score");

this.score = resultData;

}

private async getScore30() {

let resultData = await platform.getStorage("score30")

this.score30 = resultData;

}

private async getScore1min() {

let resultData = await platform.getStorage("score1min")

this.score1min = resultData;

}

private async getScore1combo() {

let resultData = await platform.getStorage("score1combo")

this.score1combo = resultData;

}

public getPerfect() {

}

public saveVecPuzzleClear() {

}

public saveScore(score: number) {

if (this.score < score) {

this.score = score;

this.setItem("score", score);

}

}

public saveScore30(score: number) {

if (this.score30 < score) {

this.score30 = score;

this.setItem("score30", score);

}

}

public saveScore1min(score: number) {

if (this.score1min < score) {

this.score1min = score;

this.setItem("score1min", score);

}

}

public saveScore1combo(score: number) {

if (this.score1combo < score) {

this.score1combo = score;

this.setItem("score1combo", score);

}

}

public saveUserName(uname: string) {

this.userName = uname;

// this.setItem("userName", uname);

}

public soundChange() {

this.sound = !this.sound;

}

public erase() {

egret.localStorage.clear();

}

public parseItem2Number(key: string): number {

let score: number = 0;

if (this.getItem(key)) {

this.score = parseInt(this.getItem(key))

}

return score;

}

public parseItem2Bool() {

}

public getItem(key: string): string {

// let resultData = platform.getStorage(key).then((result) => {

// console.log("getItem:", result);

// });

//egret.localStorage.getItem(key);

// console.log("getItem:", resultData);

// return resultData['data'];

// let resultData =await platform.getStorage(key)

return

}

private async getItem2(key: string) {

let resultData = await platform.getStorage(key)

}

public setItem(key: string, value: number) {

egret.localStorage.setItem(key, value.toString());

platform.uploadWXData(key, value);

platform.setStorage(key, value);

}

}

var SharedManager: SharedManagerClass = new SharedManagerClass();

class InputClearClass extends Input {

public constructor() {

super();

}

}

var InputClear: InputClearClass = new InputClearClass();

class InputPuzzleSelectClass extends Input {

public constructor() {

super();

}

protected mouseDownEvent(e: egret.TouchEvent): void {

// SceneManager.newScene(new SetPuzzleScene(StageManager.getStage(0)));

if (SceneManager.scene instanceof PuzzleSelectScene) {

let scene = SceneManager.scene as PuzzleSelectScene;

let p = egret.Point.create(e.stageX, e.stageY);

if (scene.stageGroup.getTransformedBounds(this).containsPoint(p)) {

let offx = e.stageX - scene.stageGroup.x;

let offy = e.stageY - scene.stageGroup.y;

offx /= (100 + 22);

offy /= (100 + 40);

let sid = 5 \* ~~offy + ~~offx;

if (sid > 24 || sid < 0) sid = 0;

SceneManager.newScene(new SetPuzzleScene(StageManager.getStage(sid)));

} else if (scene.backImg.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new SelectScene());

}

}

}

protected mouseUpEvent(param1: egret.TouchEvent): void {

}

protected mouseOutEvent(param1: egret.TouchEvent): void {

}

}

var InputPuzzleSelect: InputPuzzleSelectClass = new InputPuzzleSelectClass();

class InputRankingClass extends Input {

public constructor() {

super();

}

protected mouseDownEvent(e: egret.TouchEvent): void {

if (e.stageY >= 1000 && e.stageX >= 256 && e.stageX <= (256 + 110)) {

App.BackMenu.updateStart();

}

}

protected mouseUpEvent(e: egret.TouchEvent): void {

App.BackMenu.updateFinish();

}

}

var InputRanking: InputRankingClass = new InputRankingClass();

class InputSelectClass extends Input {

public constructor() {

super();

}

protected mouseDownEvent(e: egret.TouchEvent) {

if (SceneManager.scene instanceof SelectScene) {

let scene = SceneManager.scene as SelectScene;

let p = egret.Point.create(e.stageX, e.stageY);

if (scene.puzzleGroup.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new PuzzleSelectScene());

} else if (scene.scoreGroup.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new SetScoreScene());

} else if (scene.score30Group.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new SetScore30Scene());

} else if (scene.score1m.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new SetScore1minScene());

} else if (scene.rankGroup.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new RankingScene());

} else if (scene.optionGroup.getTransformedBounds(this).containsPoint(p)) {

// SceneManager.newScene(new OptionScene());

} else if (scene.comboGroup.getTransformedBounds(this).containsPoint(p)) {

SceneManager.newScene(new SetScore1comboScene());

}

}

}

}

var InputSelect: InputSelectClass = new InputSelectClass();

class InputSetPuzzleClass extends Input {

public field: Field;

public fieldRect: egret.Rectangle;

public blockRect: egret.Rectangle;

public inventoryPadding: number;

public constructor() {

super();

}

protected mouseDownEvent(e: egret.TouchEvent): void {

this.blockRect = new egret.Rectangle(BlockManager.x, BlockManager.y, BlockManager.width, BlockManager.height);

if (this.blockRect.containsPoint(egret.Point.create(e.stageX, e.stageY))) {

// console.log("down0");

let selectBlock = ~~((e.stageX - this.blockRect.x) / 120);

BlockManager.mouseX = e.stageX;

BlockManager.mouseY = e.stageY;

BlockManager.mouseDownInventory(selectBlock);

// console.log("down0", selectBlock);

} else if (e.stageY >= 1000 && e.stageX >= 130 && e.stageX <= (130 + 110)) {

App.BackMenu.updateStart();

} else if (e.stageY >= 1000 && e.stageX >= 410 && e.stageX <= (410 + 110)) {

BlockManager.undo();

}

}

protected mouseMoveEvent(e: egret.TouchEvent): void {

// console.log(e.stageX);

BlockManager.mouseX = e.stageX;

BlockManager.mouseY = e.stageY;

}

protected mouseUpEvent(e: egret.TouchEvent): void {

App.BackMenu.updateFinish();

if (!BlockManager.mouseDownBlock) {

return;

}

InputSetPuzzle.fieldRect = new egret.Rectangle(this.field.x, this.field.y, this.field.width, this.field.height);

if (this.fieldRect.containsPoint(egret.Point.create(e.stageX, e.stageY))) {

console.log("containsPoint")

let gx = ~~(e.stageX - this.field.x) / (120);

let gy = ~~(e.stageY - this.field.y) / (120);

/\*\*这里需要反过来，对于数组坐标系 \*/

this.field.mouseUpField(~~gy, ~~gx);

}

else {

BlockManager.mouseUp();

}

}

protected mouseOutEvent(e: egret.TouchEvent): void {

if (SceneManager.scene instanceof SetPuzzleScene)

if (this.fieldRect.containsPoint(egret.Point.create(e.stageX, e.stageY))) {

let gx = (e.stageX - this.field.x) / 120;

let gy = (e.stageY - this.field.y) / 120;

this.field.mouseUpField(gx, gy);

}

else {

BlockManager.mouseUp();

}

}

}

var InputSetPuzzle: InputSetPuzzleClass = new InputSetPuzzleClass();

class RankingScene extends Scene {

private bitmap: egret.Bitmap;

private isdisplay = true;

public constructor() {

super();

InputManager.newInput(InputRanking);

}

protected childrenCreated() {

this.init();

let backBitmap = Resource.getBlockChips(22, 220, 22, 22);

backBitmap.width = 110;

backBitmap.height = 110;

backBitmap.x = 265;

backBitmap.y = 1000;

this.addChild(backBitmap);

this.addChild(App.BackMenu);

}

private init() {

var platform: any = window.platform;

this.bitmap = platform.openDataContext.createDisplayObject(null, this.stage.stageWidth, this.stage.stageHeight);

this.addChild(this.bitmap);

platform.openDataContext.postMessage({

isDisplay: this.isdisplay,

keys: ['score', 'score30', 'score1min', 'score1combo'],

year: (new Date()).getFullYear(),

command: "getFriendCloudStorage"

});

}

public update() {

App.BackMenu.update();

}

}

class SelectScene extends Scene {

public puzzleGroup: eui.Group;

public scoreGroup: eui.Group;

public score30Group: eui.Group;

public score1m: eui.Group;

public comboGroup: eui.Group;

public rankGroup: eui.Group;

public optionGroup: eui.Group;

public puzzleLab: eui.Label;

public scoreLab: eui.Label;

public score30Lab: eui.Label;

public score1mLab: eui.Label;

public scoreComboLab: eui.Label;

public constructor() {

super();

this.skinName = "SelectSceneSkin";

}

protected childrenCreated() {

InputManager.newInput(InputSelect);

this.score1mLab.text = SharedManager.score1min.toString();

this.scoreLab.text = SharedManager.score.toString();

this.score30Lab.text = SharedManager.score30.toString();

this.scoreComboLab.text = SharedManager.score1combo.toString();

}

}

class SetPuzzleScene extends Scene {

private field: Field;

public undoBmd: eui.Image

public hudManager: HudManagerScore;

public constructor(stageData: StageData) {

super();

Status.mode = GameMode.Puzzle;

this.addChild(this.field = new Field(stageData));

let undoBitmap = Resource.getBlockChips(0, 220, 22, 22);

undoBitmap.width = 110;

undoBitmap.height = 110;

undoBitmap.x = 410;

undoBitmap.y = 1000;

this.addChild(undoBitmap);

this.undoBmd = undoBitmap;

let backBitmap = Resource.getBlockChips(22, 220, 22, 22);

backBitmap.width = 110;

backBitmap.height = 110;

backBitmap.x = 130;

backBitmap.y = 1000;

this.addChild(backBitmap);

this.addChild(BlockManager);

this.addChild(this.hudManager = new HudManagerScore());

this.addChild(EffectManager);

InputSetPuzzle.field = this.field;

InputSetPuzzle.fieldRect = new egret.Rectangle(this.field.x, this.field.y, this.field.width, this.field.height);

InputSetPuzzle.blockRect = new egret.Rectangle(BlockManager.x, BlockManager.y, BlockManager.width, BlockManager.height);

InputManager.newInput(InputSetPuzzle);

this.addChild(App.BackMenu);

this.hudManager.stageNOLab.text = stageData.stageNo.toString();

}

public update() {

BlockManager.update();

EffectManager.update();

App.BackMenu.update();

Status.update();

this.hudManager.update();

}

}

class SetScore1minScene extends Scene {

public field: Field;

public static cnt: number;

public static isFinish: Boolean = false;

public hudManager: HudManagerScore;

public constructor() {

super();

Status.mode = GameMode.Score1M;

this.addChild(this.field = new Field(new StageData("0,0,ScoreAttack,3,3,111111111")));

let backBitmap = Resource.getBlockChips(22, 220, 22, 22);

backBitmap.width = 110;

backBitmap.height = 110;

backBitmap.x = 265;

backBitmap.y = 1000;

this.addChild(backBitmap);

BlockManager.addRandomInventoryBlock(0);

BlockManager.addRandomInventoryBlock(1);

BlockManager.addRandomInventoryBlock(2);

this.addChild(BlockManager);

this.addChild(this.hudManager = new HudManagerScore());

this.addChild(App.BackMenu);

this.addChild(EffectManager);

InputManager.newInput(InputSetScore);

InputSetScore.field = this.field;

InputManager.addChild(new CountDownSprite());

}

public update() {

if (!SetScore1minScene.isFinish) {

SetScore1minScene.cnt++;

}

if (SetScore1minScene.isFinish && this.field.matchingJob == null && SetScore1minScene.cnt >= 1800) {

SetScore1minScene.isFinish = true;

BlockManager.finish(SetScore1minScene);

}

BlockManager.update();

EffectManager.update();

App.BackMenu.update();

Status.update();

this.hudManager.update();

}

}

class SetScore30Scene extends Scene {

public field: Field;

public static cnt: number;

public static isFinish: Boolean = false;

public hudManager: HudManagerScore;

public constructor() {

super();

Status.mode = GameMode.Score30;

this.addChild(this.field = new Field(new StageData("0,0,ScoreAttack,3,3,111111111")));

let backBitmap = Resource.getBlockChips(22, 220, 22, 22);

backBitmap.width = 110;

backBitmap.height = 110;

backBitmap.x = 265;

backBitmap.y = 1000;

this.addChild(backBitmap);

BlockManager.addRandomInventoryBlock(0);

BlockManager.addRandomInventoryBlock(1);

BlockManager.addRandomInventoryBlock(2);

this.addChild(BlockManager);

this.addChild(this.hudManager = new HudManagerScore());

this.addChild(App.BackMenu);

this.addChild(EffectManager);

InputManager.newInput(InputSetScore);

InputSetScore.field = this.field;

}

public update() {

BlockManager.update();

EffectManager.update();

App.BackMenu.update();

Status.update();

this.hudManager.update();

}

}

class SetScoreScene extends Scene {

public field: Field;

public hudManager: HudManagerScore;

public constructor() {

super();

Status.mode = GameMode.Score;

this.addChild(this.field = new Field(new StageData("0,0,ScoreAttack,3,3,111111111")));

let backBitmap = Resource.getBlockChips(22, 220, 22, 22);

backBitmap.width = 110;

backBitmap.height = 110;

backBitmap.x = 265;

backBitmap.y = 1000;

this.addChild(backBitmap);

BlockManager.addRandomInventoryBlock(0);

BlockManager.addRandomInventoryBlock(1);

BlockManager.addRandomInventoryBlock(2);

this.addChild(BlockManager);

this.addChild(this.hudManager = new HudManagerScore());

this.addChild(App.BackMenu);

this.addChild(EffectManager);

InputManager.newInput(InputSetScore);

InputSetScore.field = this.field;

}

public update() {

BlockManager.update();

EffectManager.update();

App.BackMenu.update();

Status.update();

this.hudManager.update();

}

}

class BackMenuBitmapClass extends eui.Component {

private cnt: number = 0;

private updateFlag: boolean = false;

public backMenuGroup: eui.Group;

public backGrogress: eui.Rect;

public menuRect: egret.Rectangle;

private toY: number = 517;

public constructor() {

super();

this.skinName = "BackMenuSkin";

}

protected childrenCreated() {

this.backMenuGroup.y = - this.toY;

}

public updateStart() {

if (this.updateFlag) {

return;

}

egret.Tween.removeTweens(this.backMenuGroup);

egret.Tween.get(this.backMenuGroup).to({ y: this.toY }, 200, egret.Ease.cubicOut);

this.cnt = 0;

this.updateFlag = true;

}

public updateFinish() {

if (!this.updateFlag) {

return;

}

egret.Tween.removeTweens(this.backMenuGroup);

egret.Tween.get(this.backMenuGroup).to({ y: - this.toY }, 200, egret.Ease.quadIn);

this.cnt = 0;

this.updateFlag = false;

}

public update() {

if (!this.updateFlag) return;

this.backGrogress.width = this.cnt \* 10;

if (this.cnt++ === 40) {

this.updateFinish();

SceneManager.newScene(new SelectScene());

}

}

}

class BitmapDisplay extends egret.Sprite {

private text: egret.TextField;

public constructor(w: number, h: number, color: number) {

super();

this.drawRect(w, h, color);

}

private drawRect(w: number, h: number, color: number) {

this.graphics.beginFill(color);

this.graphics.drawRect(0, 0, w, h);

this.graphics.endFill();

}

public drawBorderRect(param1: number, param2: number, param3: number, param4: number, param5: number, param6: number, param7: number) {

// this.graphics.drawRect()

}

public drawString(str: string, scaleX: number, scaleY: number, color: number) {

if (!this.text) this.text = new eui.Label();

this.text.text = str;

this.text.textColor = color;

this.addChild(this.text);

}

}

class Block extends eui.Component {

public static COLOR: Array<number> = [0x484CFF, 0xFF3030, 0x37FF48, 0xFFFF34, 0xFF6EFF];

public vecLayer: Array<number>;

public drawPhase: number = 8;

public gridX: number = 0;

public gridY: number = 0;

public inventoryNumber: number = 0;

public removeFlag: boolean = false;

public constructor() {

super();

this.vecLayer = new Array<number>();

}

public setLayer(...rest) {

var i: number = 0;

var length: number = ~~(rest.length);

i = 0;

while (i < length) {

this.vecLayer.push(~~rest[i]);

i++;

}

}

public removeLayer(): boolean {

this.removeFlag = false;

this.drawPhase = 3;

let resultPoint: egret.Point;

resultPoint = this.localToGlobal(this.x, this.y);

EffectManager.addEffect(new EffectLayerErase(resultPoint.x, resultPoint.y, Block.COLOR[this.vecLayer[0]]));

this.vecLayer.shift();

if (this.vecLayer.length != 0) {

this.draw();

return false;

}

this.destroy();

return true;

}

public destroy() {

BlockManager.removeBlock(this);

}

public draw() {

var i: number = 0;

var length: number = this.vecLayer.length;

if (this.drawPhase == 9) {

return;

}

this.drawPhase++;

this.removeChildren();

i = 0;

while (i < length) {

let blockChip = Resource.getBlockChips(((2 - i) \* 5 + this.drawPhase) \* 44, this.vecLayer[i] \* 44)

blockChip.width = 120;

blockChip.height = 120;

this.addChild(blockChip);

i++;

}

}

public update() {

this.draw();

}

}

class BlockManagerClass extends eui.Group {

public vecBlock: Array<Block>;

public vecInventoryBlock: Array<Block>;

public field: Field;

public mouseDownBlock: Block;

public mouseX: number = 0;

public mouseY: number = 0;

public vec2UndoString: Array<Array<string>>;

public inventoryPadding: number = 20;

public constructor() {

super();

this.tidy();

}

public reset(field: Field): void {

this.removeChildren();

this.vecBlock = new Array<Block>();

this.vecInventoryBlock = [null, null, null, null, null];

this.vec2UndoString = [];

this.field = field;

this.inventoryPadding = 0;

}

public tidy() {

// let horizontalLayout = new eui.HorizontalLayout();

// horizontalLayout.horizontalAlign = egret.HorizontalAlign.CENTER;

this.bottom = 150;

this.horizontalCenter = 0;

// this.layout = horizontalLayout;

}

public update(): void {

var i = 0;

var length = this.vecBlock.length;

i = 0;

while (i < length) {

this.vecBlock[i].update();

i++;

}

if (this.mouseDownBlock != null) {

this.mouseDownBlock.x = BlockManager.mouseX - 60;

this.mouseDownBlock.y = BlockManager.mouseY - 60;

}

}

public addBlock(block: Block): void {

this.vecBlock.push(block);

this.addChildAt(block, block.inventoryNumber);

}

public AddAllBlock(blockDataStr: Array<string>, bUndo: Boolean = true): void {

var i = 0;

var j = 0;

var splitStr: string = null;

var block: Block = null;

var xGrid = 0;

var yGrid = 0;

var l = blockDataStr.length;

if (bUndo) {

l--;

}

i = 0;

while (i < l) {

splitStr = blockDataStr[i];

xGrid = parseInt(splitStr.charAt(0));

yGrid = parseInt(splitStr.charAt(1));

block = new Block();

j = 2;

while (j < splitStr.length) {

block.vecLayer.push(parseInt(splitStr.charAt(j)));

j++;

}

/\*\*==9 可操作砖块\*\*/

if (xGrid == 9) {

this.addInventoryBlock(block, yGrid);

}

else {

this.field.getGrid(yGrid, xGrid).setBlock(block);

this.vecBlock.push(block);

}

block.draw();

i++;

}

while (this.vecInventoryBlock.length < 5) {

this.vecInventoryBlock.push(null);

}

if (bUndo) {

Status.score = parseInt(blockDataStr[i]);

Status.drawScore = Status.score;

}

}

/\*\*增加操作砖块\*\*/

public addInventoryBlock(block: Block, layerIndex: number): void {

while (this.vecInventoryBlock.length < layerIndex) {

this.vecInventoryBlock.push(null);

}

this.vecInventoryBlock[layerIndex] = block;

block.x = layerIndex \* 125;

block.inventoryNumber = layerIndex;

this.addChild(block);

}

public addRandomInventoryBlock(idx: number) {

let block = new Block();

block.inventoryNumber = idx;

this.addInventoryBlock(block, idx);

if (Status.mode == 1) {

block.setLayer(Math.random() \* 5, Math.random() \* 5, Math.random() \* 5, Math.random() \* 5);

}

else if (Status.mode == 2) {

block.setLayer(Math.random() \* 4, Math.random() \* 4, Math.random() \* 4, Math.random() \* 4);

}

else if (Status.mode == 3) {

block.setLayer(Math.random() \* 3, Math.random() \* 3, Math.random() \* 3, Math.random() \* 3);

}

else if (Status.mode == 4) {

block.setLayer(Math.random() \* 5, Math.random() \* 5, Math.random() \* 5, Math.random() \* 5);

}

block.draw();

}

public removeBlock(block: Block): void {

var i: number = 0;

var length: number = this.vecBlock.length;

i = 0;

while (i < length) {

if (block === this.vecBlock[i]) {

this.vecBlock.splice(i, 1);

block.parent && block.parent.removeChild(block);

// this.removeChild(block);

this.field.getGrid(block.gridX, block.gridY).block = null;

break;

}

i++;

}

}

public mouseDownInventory(idx: number): void {

if (this.vecInventoryBlock.length <= idx) {

return;

}

var down: Block = this.vecInventoryBlock[idx];

if (down != null) {

this.mouseDownBlock = down;

let p = this.localToGlobal(down.x, down.y);

down.x = p.x;

down.y = p.y;

this.parent.addChild(down);

}

}

public mouseUp(): void {

if (this.mouseDownBlock != null) {

let p = this.globalToLocal(this.mouseDownBlock.x, this.mouseDownBlock.y);

this.addChildAt(this.mouseDownBlock, this.mouseDownBlock.inventoryNumber);

this.mouseDownBlock.x = p.x;

this.mouseDownBlock.y = p.y;

egret.Tween.get(this.mouseDownBlock).to({ x: this.mouseDownBlock.inventoryNumber \* 125, y: 0 }, 100);

this.mouseDownBlock = null;

}

}

public undo() {

if (this.field.matchingJob || this.vec2UndoString.length <= 1) { return };

this.removeChildren();

this.vecInventoryBlock = [];

this.vecBlock = [];

this.field.resetGridBlock();

this.AddAllBlock(this.vec2UndoString[this.vec2UndoString.length - 2]);

this.vec2UndoString.pop();

if (SceneManager.scene instanceof SetPuzzleScene) {

if (this.vec2UndoString.length === 1) {

(SceneManager.scene as SetPuzzleScene).undoBmd.alpha = 0.2;

}

else {

(SceneManager.scene as SetPuzzleScene).undoBmd.alpha = 0.8;

}

}

}

public addUndoString() {

var i = 0;

var j = 0;

var str: string = null;

var block: Block = null;

var undoStr: Array<string> = [];

i = 0;

while (i < this.vecBlock.length) {

block = this.vecBlock[i];

str = "" + block.gridY + block.gridX;

j = 0;

while (j < block.vecLayer.length) {

str = str + block.vecLayer[j];

j++;

}

undoStr.push(str);

i++;

}

i = 0;

while (i < this.vecInventoryBlock.length) {

block = this.vecInventoryBlock[i];

if (block != null) {

str = "" + 9 + block.inventoryNumber;

j = 0;

while (j < block.vecLayer.length) {

str = str + block.vecLayer[j];

j++;

}

undoStr.push(str);

}

i++;

}

if (this.vec2UndoString.length != 0) {

undoStr.push("" + Status.score);

}

else {

undoStr.push("0");

}

this.vec2UndoString.push(undoStr);

if (SceneManager.scene instanceof SetPuzzleScene) {

if (this.vec2UndoString.length == 1) {

(SceneManager.scene as SetPuzzleScene).undoBmd.alpha = 0.2;

}

else {

(SceneManager.scene as SetPuzzleScene).undoBmd.alpha = 0.8;

}

}

}

public clearCheck(): boolean {

var i = 0;

var length = this.vecInventoryBlock.length;

if (this.vecBlock.length != 0) {

console.log("clearCheck false length", this.vecBlock);

return false;

}

i = 0;

while (i < length) {

if (this.vecInventoryBlock[i] != null) {

console.log("clearCheck false");

return false;

}

i++;

}

InputManager.newInput(null);

if (this.field.stageData.star3 <= Status.score) {

InputManager.addChild(new ClearSprite(true))

SharedManager.vecPuzzleClear[this.field.stageData.stageNo - 1] = 2;

}

else {

InputManager.addChild(new ClearSprite(false))

if (SharedManager.vecPuzzleClear[this.field.stageData.stageNo - 1] != 2) {

SharedManager.vecPuzzleClear[this.field.stageData.stageNo - 1] = 1;

}

}

SharedManager.saveVecPuzzleClear();

return true;

}

public finishCheck(cls: any): boolean {

console.log("finishCheck:", this.vecBlock.length, this.vecBlock);

if (this.vecBlock.length != 9) {

return false;

}

this.finish(cls);

return true;

}

public finish(cls: any): void {

Status.finishTime = new Date().getTime();

var score = Status.score;

switch (Status.mode) {

case 1:

SharedManager.saveScore(score);

break;

case 2:

SharedManager.saveScore30(score);

SetScore30Scene.isFinish = true;

break;

case 3:

SharedManager.saveScore1min(score);

SetScore1minScene.isFinish = true;

break;

case 4:

SharedManager.saveScore1combo(score);

SetScore1minScene.isFinish = true;

}

let f = new ClearSprite(true, true);

f.retryScene = cls;

InputManager.addChild(f);

}

}

var BlockManager: BlockManagerClass = new BlockManagerClass();

class ClearSprite extends eui.Component {

public clearGroup: eui.Group;

public menuGroup: eui.Group;

public retryGroup: eui.Group;

public nextGroup: eui.Group;

public nextLab: eui.Label;

public statusLab: eui.Label;

public retryScene: any;

private toY: number = 500;

private isPerfect: boolean;

private isFinish: boolean

public constructor(isPerfect: boolean, isFinish: boolean = false) {

super();

this.isPerfect = isPerfect;

this.isFinish = isFinish;

this.skinName = "ClearSpriteSkin"

}

protected childrenCreated() {

console.log("clearCheck ClearSprite");

if (!this.isPerfect) this.statusLab.text = "Clear";

if (this.isFinish) {

this.statusLab.text = "Finish";

this.nextLab.text = "Share";

}

egret.Tween.get(this.clearGroup).to({ y: this.toY }, 500, egret.Ease.quintOut).wait(100).to({ x: 66 }, 500, egret.Ease.quintOut);

egret.Tween.get(this.menuGroup).wait(600).to({ y: this.toY }, 500, egret.Ease.quintOut);

egret.Tween.get(this.retryGroup).wait(800).to({ y: this.toY }, 500, egret.Ease.quintOut);

egret.Tween.get(this.nextGroup).wait(1000).to({ y: this.toY }, 500, egret.Ease.quintOut);

this.addEventListener("touchTap", this.onTouch, this);

}

private onTouch(e: egret.TouchEvent) {

switch (e.target) {

case this.clearGroup:

break;

case this.menuGroup:

SceneManager.newScene(new SelectScene());

break;

case this.retryGroup:

if (this.isFinish) {

SceneManager.newScene(new this.retryScene());

} else {

SceneManager.newScene(new SetPuzzleScene(StageManager.getStage(-2)));

}

break;

case this.nextGroup:

if (this.isFinish) {

} else {

let stageData = StageManager.getStage(-1);

if (stageData != null) {

SceneManager.newScene(new SetPuzzleScene(stageData));

}

}

break;

}

}

}

class CountDownSprite extends eui.Group {

public constructor() {

super();

this.percentWidth = 100;

this.percentHeight = 100;

this.touchEnabled = true;

this.touchChildren = false;

this.touchThrough = false;

this.drawRect();

}

private cdLab: eui.Label;

private drawRect() {

let rect = new eui.Rect(250, 120);

rect.fillColor = 0xB0B0AF;

rect.horizontalCenter = 0;

rect.verticalCenter = 0;

this.cdLab = new eui.Label();

this.cdLab.text = "3";

this.cdLab.textColor = 0;

this.cdLab.horizontalCenter = 0;

this.cdLab.verticalCenter = 0;

this.addChild(rect);

this.addChild(this.cdLab);

egret.Tween.get(this).wait(1500).call(() => { this.cdLab.text = "2"; }).wait(1000)

.call(() => { this.cdLab.text = "1"; }).wait(1000).call(() => { this.cdLab.text = "start"; })

.to({ alpha: 0 }, 1000, egret.Ease.elasticIn).call(() => { this.parent && (this.parent.removeChild(this)) });

}

}

class Field extends eui.Group {

public static fieldX: number = 0;

public static fieldY: number = 0;

public vec2Grid: Array<Array<Grid>>;

public fieldW: number = 0;

public fieldH: number = 0;

public matchingJob: egret.Tween;

private exScore: number = 0;

public stageData: StageData;

private vecSound: Array<egret.Sound>;

public constructor(stageData: StageData) {

super();

let layout = new eui.TileLayout();

layout.horizontalGap = 22;

layout.verticalGap = 18;

// layout.verticalAlign = egret.VerticalAlign.BOTTOM;

this.layout = layout;

this.horizontalCenter = 0;

this.verticalCenter = 0;

var w = 0;

var h = 0;

var grid: Grid = null;

this.vecSound = Resource.getMusic();

this.stageData = stageData;

layout.requestedColumnCount = this.fieldW = stageData.fieldW;

layout.requestedRowCount = this.fieldH = stageData.fieldH;

if (layout.requestedColumnCount >= 5) {

layout.horizontalGap = 2;

}

Field.fieldX = this.x;

Field.fieldY = this.y;

this.vec2Grid = [];

while (h < this.fieldH) {

this.vec2Grid.push([]);

w = 0;

while (w < this.fieldW) {

grid = new Grid(h, w);

grid.type = parseInt(this.stageData.fieldData.charAt(w \* this.fieldH + h));

this.vec2Grid[h].push(grid);

if (grid.type == 1) {

grid.drawBroderRect();

}

this.addChild(grid);

w++;

}

h++;

}

console.log(this.vec2Grid)

/\*\*\*\*!!放到这里代码可读性和逻辑性变差，但是省去了在多个场景调用!!\*\*\*\*/

BlockManager.reset(this);

BlockManager.AddAllBlock(stageData.vecBlockData, false);

BlockManager.addUndoString();

}

public getGrid(x: number, y: number): Grid {

var gridX: number = x;

var gridY: number = y;

try {

return this.vec2Grid[gridX][gridY];

}

catch (e) {

return null;

}

}

/\*\*\*检查匹配 \*\*/

public matchCheck(): void {

var i = 0;

var j = 0;

var gridX = 0;

var gridY = 0;

var blockColor = 0;

var point: egret.Point = null;

var grid: Grid = null;

var block: Block = null;

var grid1: Grid = null;

var grid2: Grid = null;

var checkBlock1: Block = null;

var checkBlock2: Block = null;

var addScore = 0;

var combo = 0;

var mode = 0;

Status.combo++;

this.matchingJob = null;

var nestFlag: Boolean = false;

i = 0;

while (i < this.fieldH) {

j = 0;

while (j < this.fieldW) {

grid = this.vec2Grid[i][j];

if (grid.block != null) {

block = grid.block;

blockColor = block.vecLayer[0];

grid1 = this.getGrid(i + 1, j);

grid2 = this.getGrid(i, j + 1);

if (grid1 != null) {

checkBlock1 = grid1.block;

}

else {

checkBlock1 = null;

}

if (grid2 != null) {

checkBlock2 = grid2.block;

}

else {

checkBlock2 = null;

}

if (checkBlock1 != null && checkBlock1.vecLayer[0] == blockColor) {

checkBlock1.removeFlag = true;

block.removeFlag = true;

nestFlag = true;

}

if (checkBlock2 != null && checkBlock2.vecLayer[0] == blockColor) {

checkBlock2.removeFlag = true;

block.removeFlag = true;

nestFlag = true;

}

}

j++;

}

i++;

}

var length = BlockManager.vecBlock.length;

i = length - 1;

while (i >= 0) {

block = BlockManager.vecBlock[i];

if (block.removeFlag && block.removeLayer()) {

}

i--;

}

if (nestFlag) {

if (Status.combo < 8) {

this.vecSound[Status.combo - 1].play(0, 1);

}

else {

this.vecSound[7].play(0, 1);

}

this.matchingJob = egret.Tween.get(this).to({}, 500).call(this.matchCheck, this);

}

else {

addScore = Status.score - this.exScore;

combo = Status.combo - 1;

if (Status.maxCombo < combo) {

Status.maxCombo = combo;

}

mode = Status.mode;

if (combo > 0) {

Status.combo = Status.combo - 1;

Status.addScore();

i = 0;

while (i < 9) {

if (i != 4) {

// EffectManager.addEffect(new EffectScore("" + Status.addScore(),i / 3,i % 3));

}

else {

// EffectManager.addEffect(new EffectScore("Bonus!",i / 3,i % 3));

}

i++;

}

}

Status.combo = 0;

if (mode == GameMode.Puzzle && !BlockManager.clearCheck()) {

BlockManager.addUndoString();

}

else if (mode == GameMode.Score) {

BlockManager.finishCheck(SetScoreScene);

}

else if (mode == GameMode.Score30) {

if (SetScore30Scene.cnt == 0) {

BlockManager.finish(SetScore30Scene);

}

BlockManager.finishCheck(SetScore30Scene);

}

else if (Status.mode == GameMode.Score1M) {

BlockManager.finishCheck(SetScore1minScene);

}

else if (Status.mode == GameMode.ScoreCombo && (Status.score != 0 || BlockManager.vecBlock.length == 20)) {

BlockManager.finish(SetScore1comboScene);

}

}

}

public mouseUpField(gx: number, gy: number): void {

var gridX = gx;

var gridY = gy;

var grid: Grid = this.getGrid(gridX, gridY);

var mouseDownBlock: Block = BlockManager.mouseDownBlock;

if (!this.matchingJob && grid != null && grid.block == null && grid.type == 1) {

grid.setBlock(mouseDownBlock);

BlockManager.vecInventoryBlock[mouseDownBlock.inventoryNumber] = null;

BlockManager.vecBlock.push(mouseDownBlock);

if (Status.mode > 0) {

BlockManager.addRandomInventoryBlock(BlockManager.mouseDownBlock.inventoryNumber);

}

if (Status.mode == 2) {

SetScore30Scene.cnt--;

}

BlockManager.mouseDownBlock = null;

this.exScore = Status.score;

this.matchingJob = egret.Tween.get(this).to({}, 200).call(this.matchCheck, this);

}

else {

BlockManager.mouseUp();

}

}

public resetGridBlock(): void {

var w = 0;

var h = 0;

while (h < this.fieldH) {

w = 0;

while (w < this.fieldW) {

if (this.vec2Grid[h][w].block) {

this.vec2Grid[h][w].removeChildren();

}

this.vec2Grid[h][w].block = null;

w++;

}

h++;

}

}

}

class Game extends eui.Component {

public constructor() {

super();

this.percentHeight = 100;

this.percentWidth = 100;

this.addChild(SceneManager);

this.addChild(InputManager);

SceneManager.newScene(new SelectScene());

this.addEventListener(egret.Event.ENTER\_FRAME, this.ent, this);

// this.addChild(new ClearSprite());

// this.touchChildren = true;

}

private ent(e: egret.Event) {

SceneManager.scene.update();

}

}

class Grid extends eui.Rect {

public type: number;

public x: number;

public y: number;

public gridX: number;

public gridY: number;

public block: Block;

private sprite: egret.Sprite

public constructor(gx: number, gy: number, ) {

super();

this.gridX = gx;

this.gridY = gy;

this.width = 120;

this.height = 120;

this.fillAlpha = 0;

this.strokeWeight = 10;

this.strokeAlpha = 0;

}

public setBlock(block: Block): void {

this.block = block;

block.gridX = this.gridX;

block.gridY = this.gridY;

// let point = this.localToGlobal(this.x, this.y);

// block.x = point.x;

// block.y = point.y;

// block.x = this.parent.x + this.x;

// block.y = this.parent.y + this.y;

block.x = 0;

block.y = 0;

this.addChild(block);

}

public drawBroderRect() {

this.strokeAlpha = 1;

// this.graphics.beginFill(0x00000);

// this.graphics.lineStyle(4, 587202559);

// this.graphics.drawRect(0, 0, 40, 40);

// this.graphics.endFill();

}

}

class HudManagerScore extends eui.Component {

public stageNOLab: eui.Label;

public scoreLab: eui.Label;

public timeLab: eui.Label;

public stageLab: eui.Label;

public constructor() {

super();

this.skinName = "HudManagerScoreSkin";

}

protected childrenCreated() {

this.setGameMode();

}

public setGameMode() {

let mode = Status.mode;

if (mode === GameMode.Puzzle) {

this.timeLab.visible = false;

} else if (mode === GameMode.Score) {

this.stageLab.visible = false;

this.stageNOLab.visible = false;

} else if (mode === GameMode.Score30) {

this.stageLab.visible = false;

this.stageNOLab.visible = false;

} else if (mode === GameMode.Score1M) {

this.stageLab.visible = false;

this.stageNOLab.visible = false;

} else if (mode === GameMode.ScoreCombo) {

this.stageLab.visible = false;

this.stageNOLab.visible = false;

}

}

public update() {

this.scoreLab.text = Status.drawScore.toString();

if (Status.mode === GameMode.Score1M) {

var timeStr;

var h = 0;

var min = 0;

var s = 0;

var cnt = SetScore1minScene.cnt;

if (cnt >= 0) {

if (cnt > 1800) {

cnt = 1800;

}

timeStr = "";

h = (1800 - cnt) / 30;

min = h % 60;

s = ~~((h \* 100) % 100);

if (min < 10) {

timeStr = timeStr + "0";

}

timeStr = timeStr + ("" + ~~min + ".");

if (s < 10) {

timeStr = timeStr + "0";

}

timeStr = timeStr + ("" + ~~s);

this.timeLab.text = timeStr;

}

else {

this.timeLab.text = "60.00";

}

}

if (Status.mode === GameMode.Score30) {

this.timeLab.text = SetScore30Scene.cnt.toString();

}

}

}

class StageData {

public stageString: string;

public packNo: number = 0;

public stageNo: number = 0;

public stageName: string;

public star3: number = 0;

public star2: number = 0;

public fieldW: number = 0;

public fieldH: number = 0;

public fieldData: string;

public vecBlockData: Array<string>;

public backGroundColor: number = 0;

public constructor(strData: string) {

this.stageString = strData;

this.convert();

}

private convert() {

var splitStr: Array<any> = null;

splitStr = this.stageString.split(",");

this.star3 = ~~(parseInt(splitStr[0]));

this.stageNo = ~~(parseInt(splitStr[1]));

this.stageName = splitStr[2];

this.fieldW = ~~(parseInt(splitStr[3]));

this.fieldH = ~~(parseInt(splitStr[4]));

this.fieldData = splitStr[5];

this.vecBlockData = new Array<string>();

var blockDataId: number = 6;

while (blockDataId < splitStr.length) {

this.vecBlockData.push(splitStr[blockDataId]);

blockDataId++;

}

}

}

class StageManagerClass {

public vecStageData: Array<StageData>;

public index: number = 0;

public constructor() {

this.vecStageData = new Array<StageData>();

this.vecStageData.push(new StageData("20,1,testes,2,1,11,900,910"));

this.vecStageData.push(new StageData("30,2,testes,3,1,111,901,911,921"));

this.vecStageData.push(new StageData("60,3,testes,3,1,111,900,911,9201"));

this.vecStageData.push(new StageData("50,4,testes,3,1,111,000,900,911,9210"));

this.vecStageData.push(new StageData("220,5,testes,3,2,111011,010,1000,21000,900,9100,92000"));

this.vecStageData.push(new StageData("140,6,tes,3,3,010111010,101,212,90001,91002,9201,9302"));

this.vecStageData.push(new StageData("400,7,score400,4,4,1111111111111111,0001,2010,1101,3110,0201,2210,1301,3310,9001,9110,9201,9310"));

this.vecStageData.push(new StageData("530,8,testes,3,3,010111010,1001,1212,0123,2130,900,9101,92012,930123"));

this.vecStageData.push(new StageData("300,9,testes,2,1,11,000,9004,91044,920444,934440,944404"));

this.vecStageData.push(new StageData("380,10,testes,2,2,1111,9004,91241,92203,932031"));

this.vecStageData.push(new StageData("380,11,testes,2,2,1111,9021,91213,92201,93012,940132"));

this.vecStageData.push(new StageData("23180,12,score400,3,4,111110011111,90041,020112,931223,132334,923440,224001,940112,201223,912334,00344"));

this.vecStageData.push(new StageData("530,13,tes,5,1,11111,00024,104022,302041,40440,9031,9142,9240,93403,942404"));

this.vecStageData.push(new StageData("1260,14,testes,5,1,11111,900,9131,92102,933201,942120"));

this.vecStageData.push(new StageData("760,15,tes,3,3,100111100,00424,20002,11313,90042,91432,92130,93420,94340"));

this.vecStageData.push(new StageData("590,16,testes,3,1,111,90230,91120,922030,931201,942120"));

this.vecStageData.push(new StageData("640,17,tes,2,2,1111,004312,110343,90324,91340,92401,93312,94024"));

this.vecStageData.push(new StageData("1290,18,1290,5,1,11111,9012,91113,921132,931123,941312"));

this.vecStageData.push(new StageData("1580,19,score400,3,3,001011111,9113,11324,942134,932430,92403,9024"));

this.vecStageData.push(new StageData("280,20,tes,5,1,11111,00232,20332,40343,9034,9123,9232,9331,9421"));

this.vecStageData.push(new StageData("2070,21,testes,4,4,1111111111111111,0012,1031,2023,3034,3114,3221,3312,2321,1314,0342,0232,0122,9013,9121,9234,9342"));

this.vecStageData.push(new StageData("1380,22,score400,3,3,101111101,101423,123241,9023,9131,9241,9342,94213"));

this.vecStageData.push(new StageData("2460,23,testes,3,3,111111111,001002,02312,20210,110301,220331,900202,910033,921202,933210,942202"));

this.vecStageData.push(new StageData("1770,24,tes,3,3,110111011,00411,01301,21021,22101,90132,91323,92241,93013,94320"));

this.vecStageData.push(new StageData("2780,25,testes,3,3,111111111,100324,014130,123424,211214,904213,911334,922124,933240,940144"));

}

public getStage(stageId: number): StageData {

if (stageId == -1) {

if (this.vecStageData.length > this.index + 1) {

this.index++;

return this.vecStageData[this.index];

}

return null;

}

if (stageId == -2) {

return this.vecStageData[this.index];

}

this.index = stageId;

return this.vecStageData[this.index];

}

}

var StageManager: StageManagerClass = new StageManagerClass();

class StatusClass {

public score: number = 0;

public drawScore: number = 0;

public combo: number = 0;

public maxCombo: number = 0;

public mode: number = 0;

public startTime: number = 0;

public finishTime: number = 0;

private vecComboScore: Array<number> = [0, 10, 20, 40, 70, 110, 160, 220, 290, 370, 460, 560, 670, 790, 920, 1060, 1210, 1370, 1540, 1720, 1910, 2110, 2320, 2540, 2770, 3010, 3260, 3520, 3790, 4070, 4360, 4660, 4970, 5290, 5620, 5960, 6310, 6670, 7040, 7420, 7810, 8210, 8620, 9040, 9470, 9910, 10360, 10820, 11290, 11770];

public constructor() {

}

public reset(mode: number) {

this.mode = mode;

this.combo = 0;

this.maxCombo = this.vecComboScore.length;

this.score = 0;

this.drawScore = 0;

this.startTime = new Date().getDate();

this.finishTime = 0;

}

public addScore(): number {

let add = this.vecComboScore[this.combo];

this.score += add;

return add;

}

public getPlayTime(): string {

let offTime = this.finishTime - this.startTime;

return;

}

public update() {

if (this.drawScore != this.score) {

this.drawScore = this.drawScore + (~~((this.score - this.drawScore) / 5) + 1);

}

}

}

var Status: StatusClass = new StatusClass();

enum GameMode {

Puzzle,

Score,

Score30,

Score1M,

ScoreCombo

}

class App {

public constructor() {

}

public static BackMenu: BackMenuBitmapClass;

public static startUp(mainCotent: eui.UILayer) {

console.log("startUp");

// SharedManager = new SharedManagerClass();

let back = new eui.Rect();

back.fillColor = 0x888888;

back.percentWidth = 100;

back.percentHeight = 100;

mainCotent.addChild(back);

App.BackMenu = new BackMenuBitmapClass();

mainCotent.addChild(new Game());

}

}

//////////////////////////////////////////////////////////////////////////////////////

//

// Copyright (c) 2014-present, Egret Technology.

// All rights reserved.

// Redistribution and use in source and binary forms, with or without

// modification, are permitted provided that the following conditions are met:

//

// \* Redistributions of source code must retain the above copyright

// notice, this list of conditions and the following disclaimer.

// \* Redistributions in binary form must reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the

// documentation and/or other materials provided with the distribution.

// \* Neither the name of the Egret nor the

// names of its contributors may be used to endorse or promote products

// derived from this software without specific prior written permission.

//

// THIS SOFTWARE IS PROVIDED BY EGRET AND CONTRIBUTORS "AS IS" AND ANY EXPRESS

// OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES

// OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

// IN NO EVENT SHALL EGRET AND CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA,

// OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF

// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

// EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

//

//////////////////////////////////////////////////////////////////////////////////////

class Main extends eui.UILayer {

protected createChildren(): void {

super.createChildren();

egret.lifecycle.addLifecycleListener((context) => {

// custom lifecycle plugin

})

egret.lifecycle.onPause = () => {

egret.ticker.pause();

}

egret.lifecycle.onResume = () => {

egret.ticker.resume();

}

//inject the custom material parser

//注入自定义的素材解析器

let assetAdapter = new AssetAdapter();

egret.registerImplementation("eui.IAssetAdapter", assetAdapter);

egret.registerImplementation("eui.IThemeAdapter", new ThemeAdapter());

this.runGame().catch(e => {

console.log(e);

})

}

private async runGame() {

await this.loadResource();

await platform.login();

const userInfo = await platform.getUserInfo();

console.log(userInfo);

await SharedManager.init();

this.createGameScene();

}

private async loadResource() {

console.log("loadResource");

try {

// const loadingView = new LoadingUI();

// this.stage.addChild(loadingView);

await RES.loadConfig("resource/default.res.json", "resource/");

await this.loadTheme();

await RES.loadGroup("preload", 0);

// this.stage.removeChild(loadingView);

}

catch (e) {

console.error(e);

}

}

private loadTheme() {

return new Promise((resolve, reject) => {

// load skin theme configuration file, you can manually modify the file. And replace the default skin.

//加载皮肤主题配置文件,可以手动修改这个文件。替换默认皮肤。

let theme = new eui.Theme("resource/default.thm.json", this.stage);

theme.addEventListener(eui.UIEvent.COMPLETE, () => {

resolve();

}, this);

})

}

private textfield: egret.TextField;

/\*\*

\* 创建场景界面

\* Create scene interface

\*/

protected createGameScene(): void {

App.startUp(this);

}

}