# **教你利用工具将单机改造成对战网游（详细教程）**

作者：[Matchvs](http://gad.qq.com/user/index?id=2257559)

链接：<http://gad.qq.com/article/detail/48046>

前言：本Demo原来是Cocos Creator官方的一个Demo，本文章利用了第三方联网插件工具[Matchvs](http://www.matchvs.com/home)将其改造成了一个三人对战的Demo，（[在线体验地址](http://cocos.matchvs.com/cocos-matchvs-demo/web-desktop/)）。

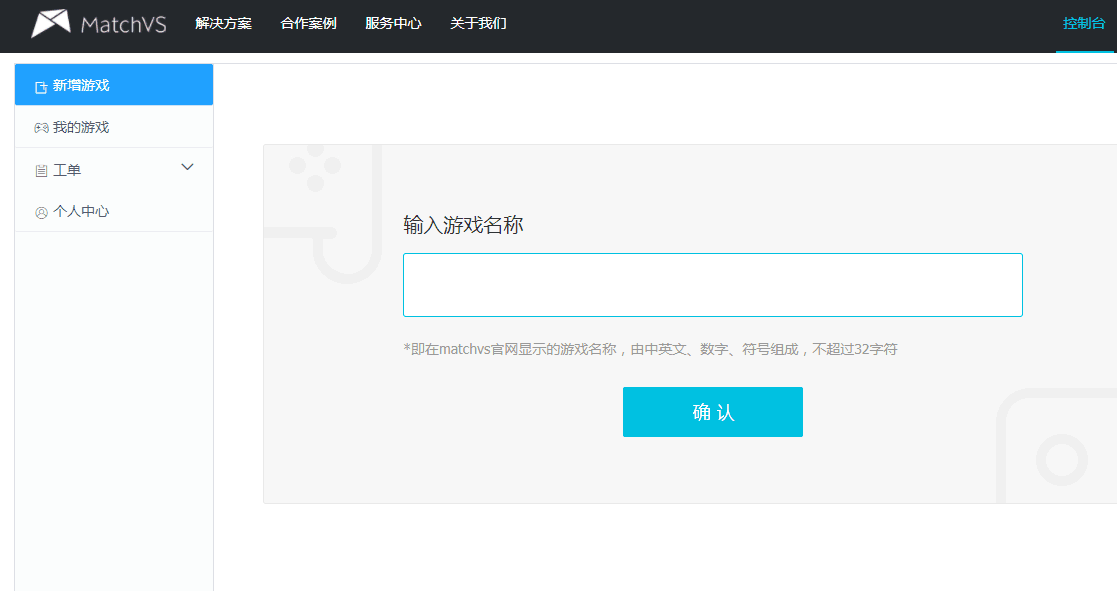
注意：

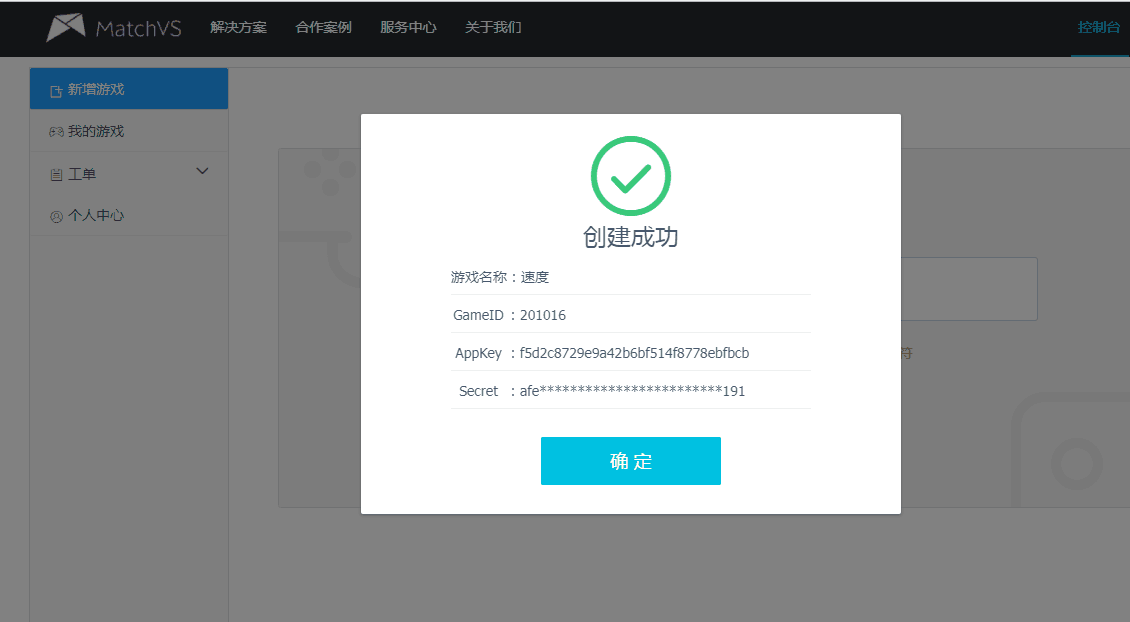
1.游戏满三人才可以开启，匹配成功后，玩家通过键盘AD键操纵小怪物向左向右移动抢摘星星。

2.下载Demo源码后，需用Cocos Creator打开工程（建议使用1.7.0及以上版本）。

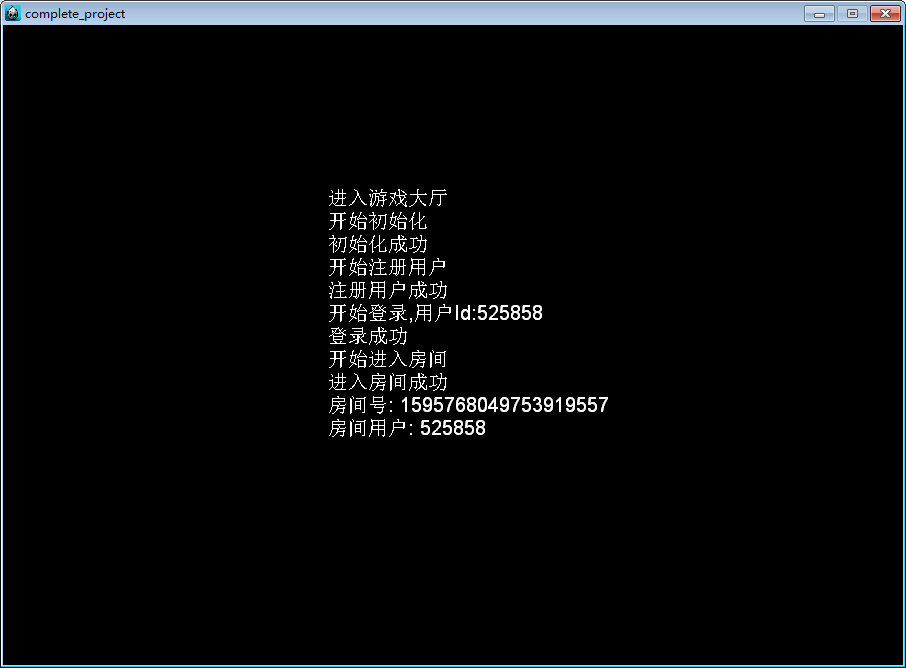
## 游戏配置

Demo运行之前需要去[Matchvs 官网](http://www.matchvs.com/)配置游戏相关信息，以获取Demo运行所需要的GameID、AppKey、SecretID。如图：





获取到相关游戏信息之后，运行Demo，即可进入房间，准备开始游戏，如图所示：



## 初始化SDK

在引入SDK之后，在初始化前需要先调用Matchvs.MatchvsEngine.getInstance()获取一个Matchvs引擎对象实例：

var engine = Matchvs.MatchvsEngine.getInstance();

另外我们需要定义一个对象，该对象定义一些回调方法，用于获取游戏中玩家加入、离开房间、数据收发的信息，这些方法在特定的时刻会被SDK调用。

var response = ｛

// 可以现在定义一些回调方法，也可以过后再定义。

｝;

为方便使用，我们把engine和reponse放到单独的文件Mvs.js中，使用module.exports将它们作为全局变量使用：

var engine = Matchvs.MatchvsEngine.getInstance();

var response = ｛｝;

module.exports = ｛

engine: engine,

response: engine

｝;

// 文件路径：assets\scripts\Mvs.js

其他文件可以用require函数引入engine和reponse：

var mvs = require("Mvs");

// 引擎实例：mvs.engine

// 引擎回调实现：mvs.response

完成以上步骤后，我们可以调用初始化接口建立相关资源。

mvs.engine.init(response, channel, platform, gameId);

// 文件路径：assets\scripts\Lobby.js

**注意** 在整个应用全局，开发者只需要对引擎做一次初始化。

## 建立连接

接下来，我们就可以从Matchvs获取一个合法的用户ID，通过该ID连接至Matchvs服务端。

获取用户ID：

cc.Class(｛

onLoad: function() ｛

mvs.response.registerUserResponse = this.registerUserResponse.bind(this);

mvs.engine.registerUser();

｝,

registerUserResponse: function(userInfo) ｛

// 注册成功，userInfo包含相关用户信息

｝,

// ...

｝)

// 文件路径：assets\scripts\Lobby.js

用户信息需要保存起来，我们使用一个类型为对象的全局变量GLB来存储：

GLB.userInfo = userInfo;

登录：

cc.Class(｛

onLoad: function() ｛

// ...

mvs.engine.login(userInfo.id, userInfo.token, gameId, gameVersion, appKey,

secret, deviceId, gatewayId);

// ...

｝,

loginResponse: function(loginRsp) ｛

// 登录成功，loginRsp包含登录相关信息

｝,

// ...

｝)

// 文件路径：assets\scripts\Lobby.js

## 加入房间

成功连接至Matchvs后，立即随机匹配加入一个房间进行游戏。

代码如下:

cc.Class(｛

loginResponse: function() ｛

// ...

mvs.response.joinRoomResponse = this.joinRoomResponse.bind(this);

mvs.engine.joinRandomRoom(maxPlayer, userProfile);

// ...

｝,

joinRoomResponse: function(status, userInfoList, roomInfo) ｛

// 加入房间成功，status表示结果，roomUserInfoList为房间用户列表，roomInfo为房间信息

// ...

｝,

// ...

｝)

// 文件路径：assets\scripts\Lobby.js

## 停止加入

我们设定如果有3个玩家匹配成功则满足开始条件且游戏设计中不提供中途加入，此时需告诉Matchvs不要再向房间里加人。

代码如下:

cc.Class(｛

joinRoomResponse: function(status, userInfoList, roomInfo) ｛

// 加入房间成功，status表示结果，roomUserInfoList为房间用户列表，roomInfo为房间信息

// ...

if (userIds.length >= GLB.MAX\_PLAYER\_COUNT) ｛

mvs.response.joinOverResponse = this.joinOverResponse.bind(this); // 关闭房间之后的回调

var result = mvs.engine.joinOver("");

this.labelLog("发出关闭房间的通知");

if (result !== 0) ｛

this.labelLog("关闭房间失败，错误码：", result);

｝

GLB.playerUserIds = userIds;

｝

｝,

joinOverResponse: function(joinOverRsp) ｛

if (joinOverRsp.status === 200) ｛

this.labelLog("关闭房间成功");

// ...

｝ else ｛

this.labelLog("关闭房间失败，回调通知错误码：", joinOverRsp.status);

｝

｝,

｝)

// 文件路径：assets\scripts\Lobby.js

在这里需要记下房间的用户列表，记入到全局变量GLB.playerUserIds中，后面要使用到。

## 发出游戏开始通知

如果收到服务端的房间关闭成功的消息，就可以通知游戏开始了。

cc.Class(｛

// ...

joinOverResponse: function(joinOverRsp) ｛

if (joinOverRsp.status === 200) ｛

this.labelLog("关闭房间成功");

this.notifyGameStart();

｝ else ｛

this.labelLog("关闭房间失败，回调通知错误码：", joinOverRsp.status);

｝

｝,

notifyGameStart: function () ｛

GLB.isRoomOwner = true;

var event = ｛

action: GLB.GAME\_START\_EVENT,

userIds: GLB.playerUserIds

｝

mvs.response.sendEventResponse = this.sendEventResponse.bind(this); // 设置事件发射之后的回调

mvs.response.sendEventNotify = this.sendEventNotify.bind(this); // 设置事件接收的回调

var result = mvs.engine.sendEvent(JSON.stringify(event));

// ...

// 发送的事件要缓存起来，收到异步回调时用于判断是哪个事件发送成功

GLB.events[result.sequence] = event;

｝,

sendEventResponse: function (info) ｛

// ... 输入校验

var event = GLB.events[info.sequence]

if (event && event.action === GLB.GAME\_START\_EVENT) ｛

delete GLB.events[info.sequence]

this.startGame()

｝

｝,

sendEventNotify: function (info) ｛

if (info

&& info.cpProto

&& info.cpProto.indexOf(GLB.GAME\_START\_EVENT) >= 0) ｛

GLB.playerUserIds = [GLB.userInfo.id]

// 通过游戏开始的玩家会把userIds传过来，这里找出所有除本玩家之外的用户ID，

// 添加到全局变量playerUserIds中

JSON.parse(info.cpProto).userIds.forEach(function(userId) ｛

if (userId !== GLB.userInfo.id) GLB.playerUserIds.push(userId)

｝);

this.startGame()

｝

｝,

startGame: function () ｛

this.labelLog('游戏即将开始')

cc.director.loadScene('game')

｝,

｝)

// 文件路径：assets\scripts\Lobby.js

## 游戏数据传输

游戏进行中在创建星星、玩家进行向左、向右操作时，我们将这些操作广播给房间内其他玩家。界面上同步展示各个玩家的状态变化。

其中星星是房主创建和展示，然后通知其他玩家，其他玩家收到消息后展示，相关的代码如下：

cc.Class(｛

onLoad: function() ｛

mvs.response.sendEventNotify = this.sendEventNotify.bind(this);

// ...

｝,

sendEventNotify: function (info) ｛

// ...

if (info.cpProto.indexOf(GLB.NEW\_START\_EVENT) >= 0) ｛

// 收到创建星星的消息通知，则根据消息给的坐标创建星星

this.createStarNode(JSON.parse(info.cpProto).position)

｝ /\* 其他else if条件 \*/

｝,

// 根据坐标位置创建渲染星星节点

createStarNode: function (position) ｛

// ...

｝,

// 发送创建星星事件

spawnNewStar: function () ｛

if (!GLB.isRoomOwner) return; // 只有房主可创建星星

var event = ｛

action: GLB.NEW\_START\_EVENT,

position: this.getNewStarPosition()

｝

var result = mvs.engine.sendEvent(JSON.stringify(event))

if (!result || result.result !== 0)

return console.error('创建星星事件发送失败');

this.createStarNode(event.position);

｝,

// 随机返回'新的星星'的位置

getNewStarPosition: function () ｛

// ...

｝,

// ...

｝)

// 文件路径：assets\scripts\Game.js

玩家进行向左、向右操作时，这些消息会发送给其他玩家：

cc.Class(｛

setInputControl: function () ｛

var self = this;

cc.eventManager.addListener(｛

event: cc.EventListener.KEYBOARD,

onKeyPressed: function (keyCode, event) ｛

var msg = ｛ action: GLB.PLAYER\_MOVE\_EVENT ｝;

switch (keyCode) ｛

case cc.KEY.a:

case cc.KEY.left:

msg.accLeft = true;

msg.accRight = false;

break;

case cc.KEY.d:

case cc.KEY.right:

msg.accLeft = false;

msg.accRight = true;

break;

default:

return;

｝

var result = mvs.engine.sendEvent(JSON.stringify(msg));

if (result.result !== 0)

return console.error("移动事件发送失败");

self.accLeft = msg.accLeft;

self.accRight = msg.accRight;

｝,

onKeyReleased: function (keyCode, event) ｛

var msg = ｛ action: GLB.PLAYER\_MOVE\_EVENT ｝;

switch (keyCode) ｛

case cc.KEY.a:

msg.accLeft = false;

break;

case cc.KEY.d:

msg.accRight = false;

break;

default:

return;

｝

var result = mvs.engine.sendEvent(JSON.stringify(msg));

if (result.result !== 0)

return console.error("停止移动事件发送失败");

if (msg.accLeft !== undefined) self.accLeft = false;

if (msg.accRight !== undefined) self.accRight = false;

｝

｝, self.node);

｝,

onLoad: function () ｛

// ...

this.setInputControl();

｝

// ...

｝)

// 文件路径：assets\scripts\Player1.js

cc.Class(｛

sendEventNotify: function (info) ｛

if (/\* ... \*/) ｛

// ...

｝ else if (info.cpProto.indexOf(GLB.PLAYER\_MOVE\_EVENT) >= 0) ｛

// 收到其他玩家移动的消息，根据消息信息修改加速度

this.updatePlayerMoveDirection(info.srcUserId, JSON.parse(info.cpProto))

｝ /\* 更多else if条件\*/

｝,

// 更新每个玩家的移动方向

updatePlayerMoveDirection: function (userId, event) ｛

// ...

｝,

// ...

｝)

// 文件路径：assets\scripts\Game.js

考虑到数据同步会有延迟，不同客户端收到的数据的延迟也会有差异，如果只在同步玩家左右移动的操作数据，那么过一段时间之后，不同客户端的小怪物位置可能会不一样，因此每隔一段时间还是需要再同步一次小怪物的位置、速度和加速度数据：

cc.Class(｛

onLoad: function () ｛

// ...

setInterval(() => ｛

mvs.engine.sendEvent(JSON.stringify(｛

action: GLB.PLAYER\_POSITION\_EVENT,

x: this.node.x,

xSpeed: this.xSpeed,

accLeft: this.accLeft,

accRight: this.accRight,

ts: new Date().getTime()

｝));

｝, 200);

// ..

｝

// ...

｝)

// 文件路径：assets\scripts\Player1.js

cc.Class(｛

sendEventNotify: function (info) ｛

if (/\* ... \*/) ｛

// ...

｝ else if (info.cpProto.indexOf(GLB.PLAYER\_POSITION\_EVENT) >= 0) ｛

// 收到其他玩家的位置速度加速度信息，根据消息中的值更新状态

this.receiveCountValue++;

this.receiveCount.string = "receive msg count: " + this.receiveCountValue;

var cpProto = JSON.parse(info.cpProto);

var player = this.getPlayerByUserId(info.srcUserId);

if (player) ｛

player.node.x = cpProto.x;

player.xSpeed = cpProto.xSpeed;

player.accLeft = cpProto.accLeft;

player.accRight = cpProto.accRight;

｝

// ...

｝ /\* 更多else if条件 \*/

｝,

// ...

｝)

// 文件路径：assets\scripts\Game.js

最终效果如下：



搞定。