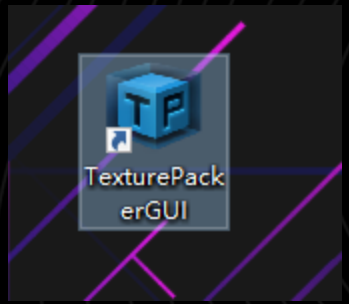
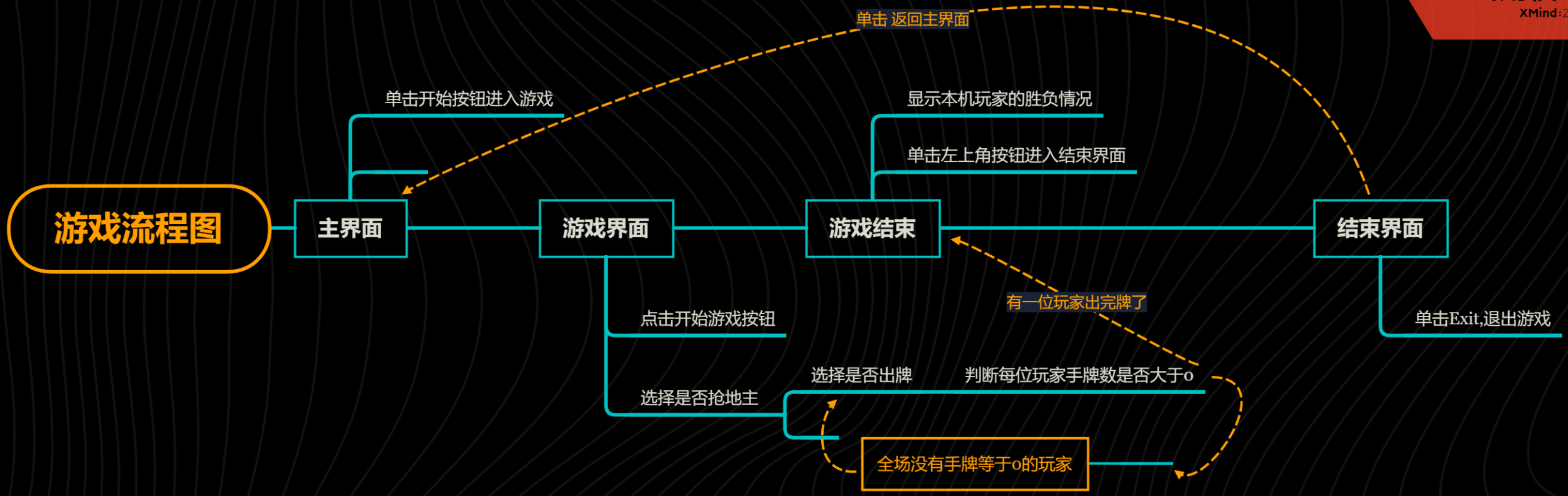
基于Cocos2dx-4.0的斗地主

计创18-连月菡

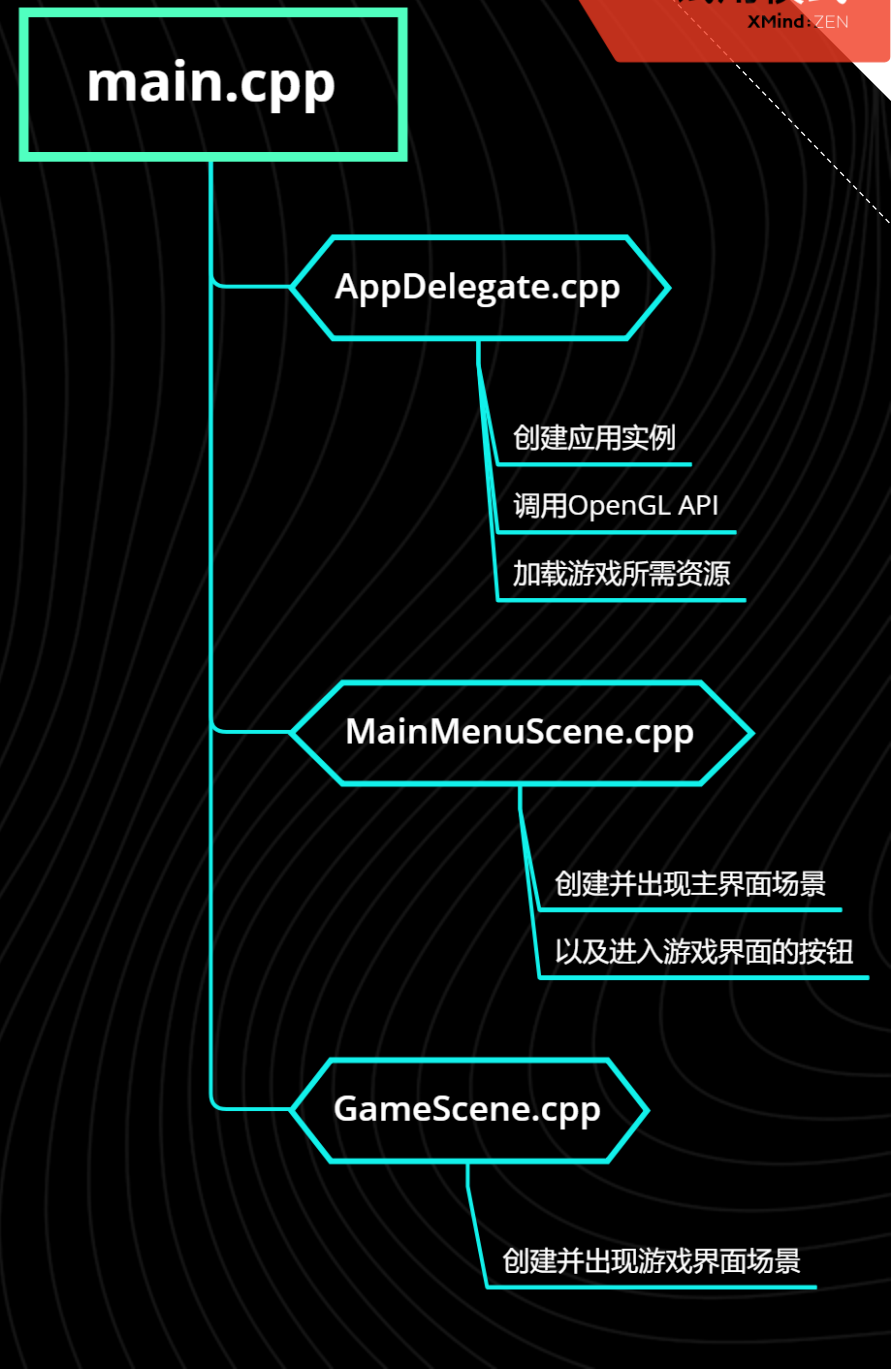
1.3 为什么选择 Cocos2d-x

Cocos2d 家族有许多版本，那么在这些版本中，我们为什么选择 Cocos2d-x 呢？因为 Cocos2d-x 不但具有 Cocos2d 引擎的功能，它自身还具有很多优势。前面说过，Cocos2d-x 是使用 C++语言开发的，可以在运行在 Windows、iOS、Android 平台上，这是第 1 个优势；第 2 个优势是 Cocos2d-x 是国内团队开发的，虽然 Cocos2d-x 官网语言为英语，但国内还有很多中文社区；第 3 个优势是 Cocos2d-x 引擎占国内手机游戏开发的使用份额接近 70%，而占国外的使用份额则接近 25%。在苹果 App 排行榜 TOP 10 里面，有 7 款 App 都是用 Cocos2d-x 引擎开发的。当前国内大多数公司选择使用 Cocos2d-x 开发 2D 游戏。使用 Cocos2d-x 引擎可以让你的游戏支持更多平台；第 4 个优势是 Cocos2d-x 有很多辅助开发工具，比如地图制作工具 Tiled、纹理生成工具 TexturePacker、CocoStudio（界面设计工具）等，便于开发。

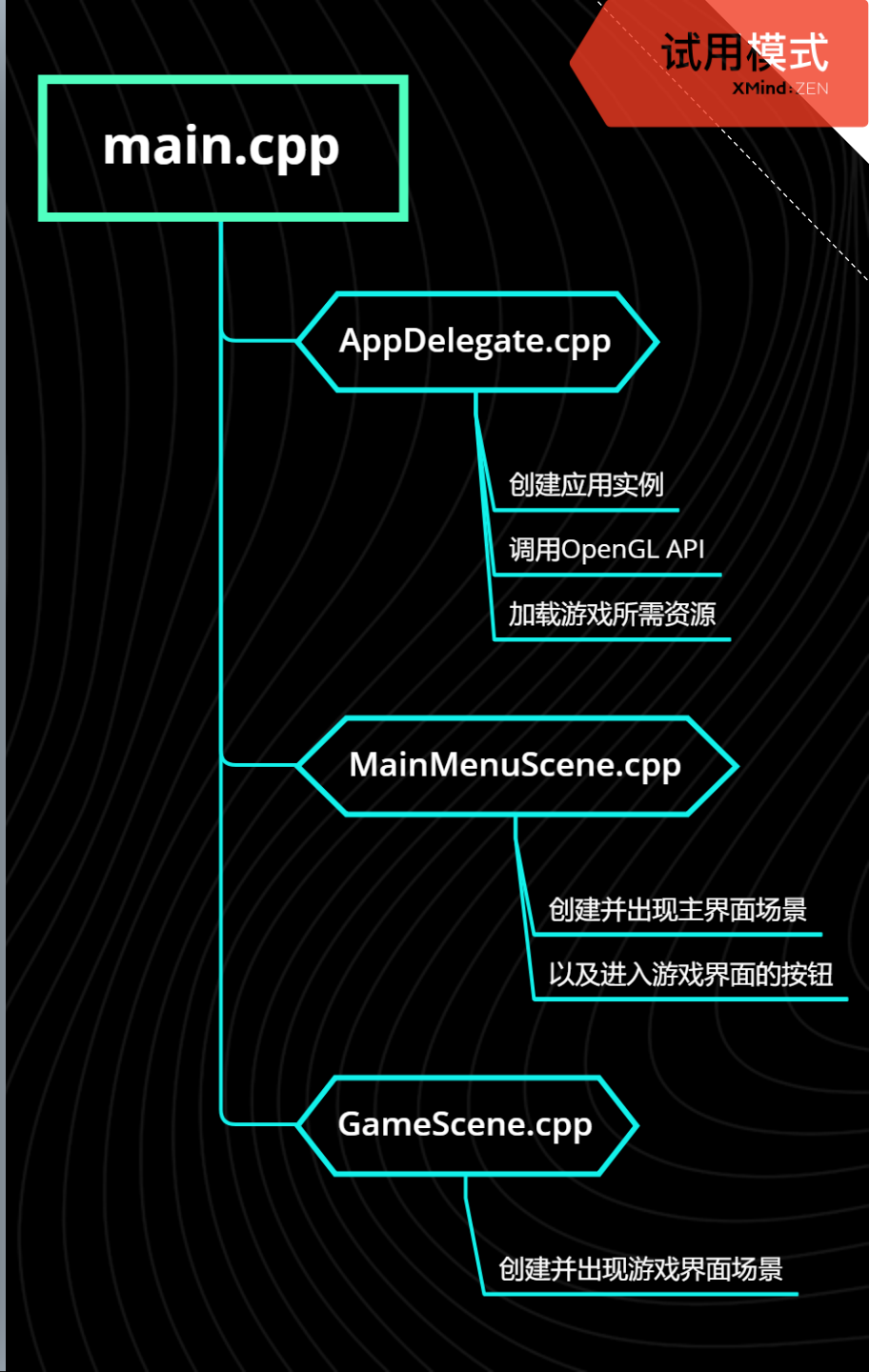




```cpp
//main.cpp
int WINAPI _tWinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPTSTR    lpCmdLine,
                     int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    AppDelegate app;
    return Application::getInstance()->run();
}

//AppDelegate.cpp
bool AppDelegate::applicationDidFinishLaunching() {
    auto director = Director::getInstance();
    auto glview = director->getOpenGLView();
    if(!glview) {
        glview = GLViewImpl::createWithRect("Fight Against Landlord ver1.0.0", cocos2d::Rect(0, 0, designResolutionSize.width, designResolutionSize.height));
#else
        glview = GLViewImpl::create("斗地主");
#endif
        director->setOpenGLView(glview);
    }
    // Set the design resolution
    glview->setDesignResolutionSize(designResolutionSize.width, designResolutionSize.height, ResolutionPolicy::SHOW_ALL);
    auto frameSize = glview->getFrameSize();
    // 加载游戏需要资源
    auto frameCache = SpriteFrameCache::getInstance();
    frameCache->addSpriteFramesWithFile("poker_b.plist", "poker_b.png");
    frameCache->addSpriteFramesWithFile("gameover/ddzsingle_map_lvl.plist", "gameover/ddzsingle_map_lvl.png");
    frameCache->addSpriteFramesWithFile("gameover/nt_result_base.plist", "gameover/nt_result_base.png");

    auto scene = MainMenu::createScene();
    director->runWithScene(scene);
    return true;
}
```

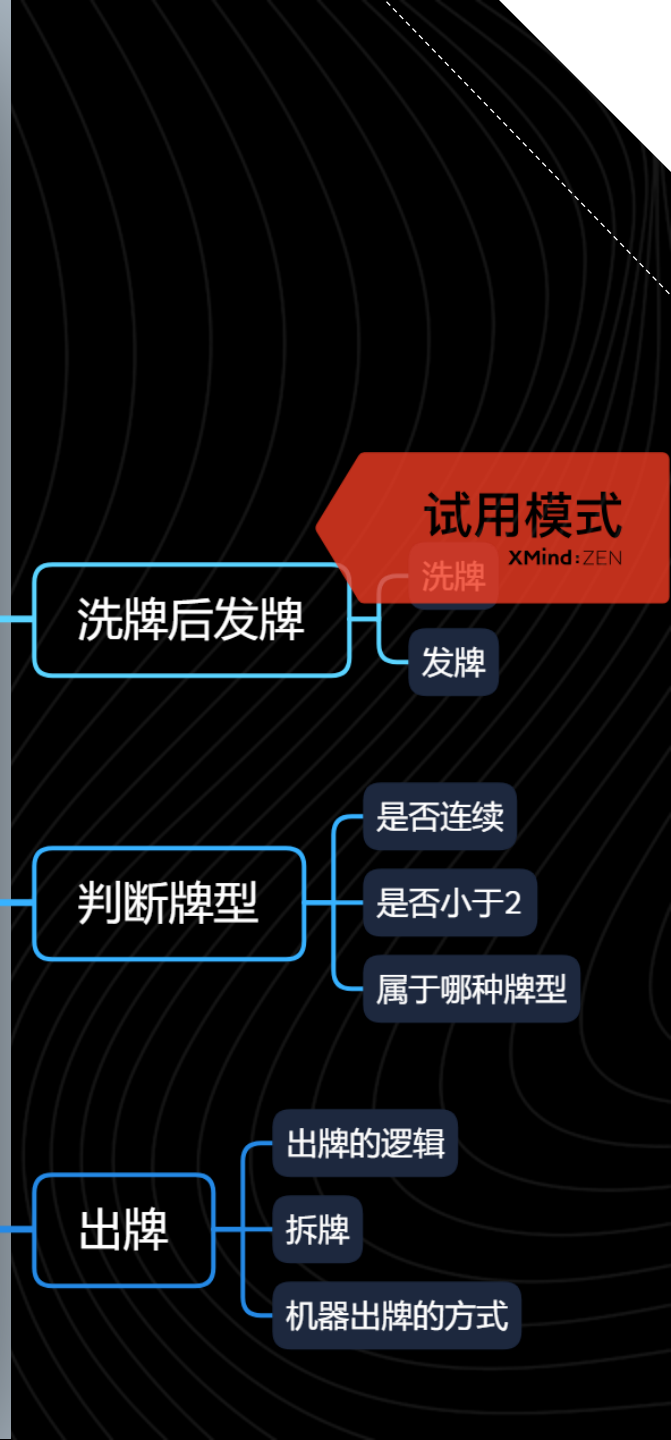
```cpp
//MainMenuScene.cpp
Scene* MainMenu::createScene()
{
    auto layer=MainMenu::create();
    auto scene = Scene::create();
    scene->addChild(layer);
    return scene;
}

bool MainMenu::init()
{
    if (!Layer::init())
    {
        return false;
    }
    auto visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    //主菜单 标题，开始游戏
    auto menuTitle = MenuItemImage::create("MenuPic.png", "MenuPic.png");
    auto playItem=MenuItemImage::create("CloseNormal.png","CloseNormal.png",CC_CALLBACK_1(MainMenu::GoToGameScene,this));
    auto menu = Menu::create(menuTitle, playItem, NULL);
    menu->alignItemsVerticallyWithPadding(visibleSize.height/9);//item的排列方式
    this->addChild(menu);
    return true;
}

void MainMenu::GoToGameScene(Ref* pSender)//切换到游戏场景
{
    auto scene = GameScene::createScene();
    Director::getInstance()->replaceScene(scene);
}

```



GameScene.cpp

```cpp
bool GameScene::init(){
  srand(time(0));//初始化玩家信息
  auto name_list = FileUtils::getInstance()->getValueMapFromFile("strings.xml").at("name_list").asValueVector();
  if (!s_runtimeData._isReady)// 如果是首次进入此处，创建人物信息
  {
      int name_index_1 = rand() % name_list.size();
      int name_index_2 = rand() % name_list.size();
      int name_index_3 = rand() % name_list.size();

      while ((name_index_2 == name_index_1 )|| (name_index_2 == name_index_3)|| (name_index_1 == name_index_3))
      {
          name_index_1 = rand() % name_list.size();
         name_index_2 = rand() % name_list.size();
          name_index_3 = rand() % name_list.size();
      }

      s_runtimeData._playerinfo1._name = name_list[name_index_1].asString();
      s_runtimeData._playerinfo2._name = name_list[name_index_2].asString();
      s_runtimeData._playerinfo3._name = name_list[name_index_3].asString();
      s_runtimeData._isReady = true;
  }
  _player = Player::create(s_runtimeData._playerinfo._name, true);
}


void GameScene::initCards()//初始化卡牌信息
{
    PokerInfo info;
    for (int i = 0; i < 13; i++)
        for (int j = 0; j < 4; j++)
        {
            info._num = (PokerNum)i;
            info._tag = (PokerTag)j;
            _pokeInfo.push_back(info);
        }
    info._num = (PokerNum)13;
    info._tag = (PokerTag)0;
    _pokeInfo.push_back(info);
    info._num = (PokerNum)14;
    info._tag = (PokerTag)0;
    _pokeInfo.push_back(info);
}

```

初始化玩家信息和卡牌堆



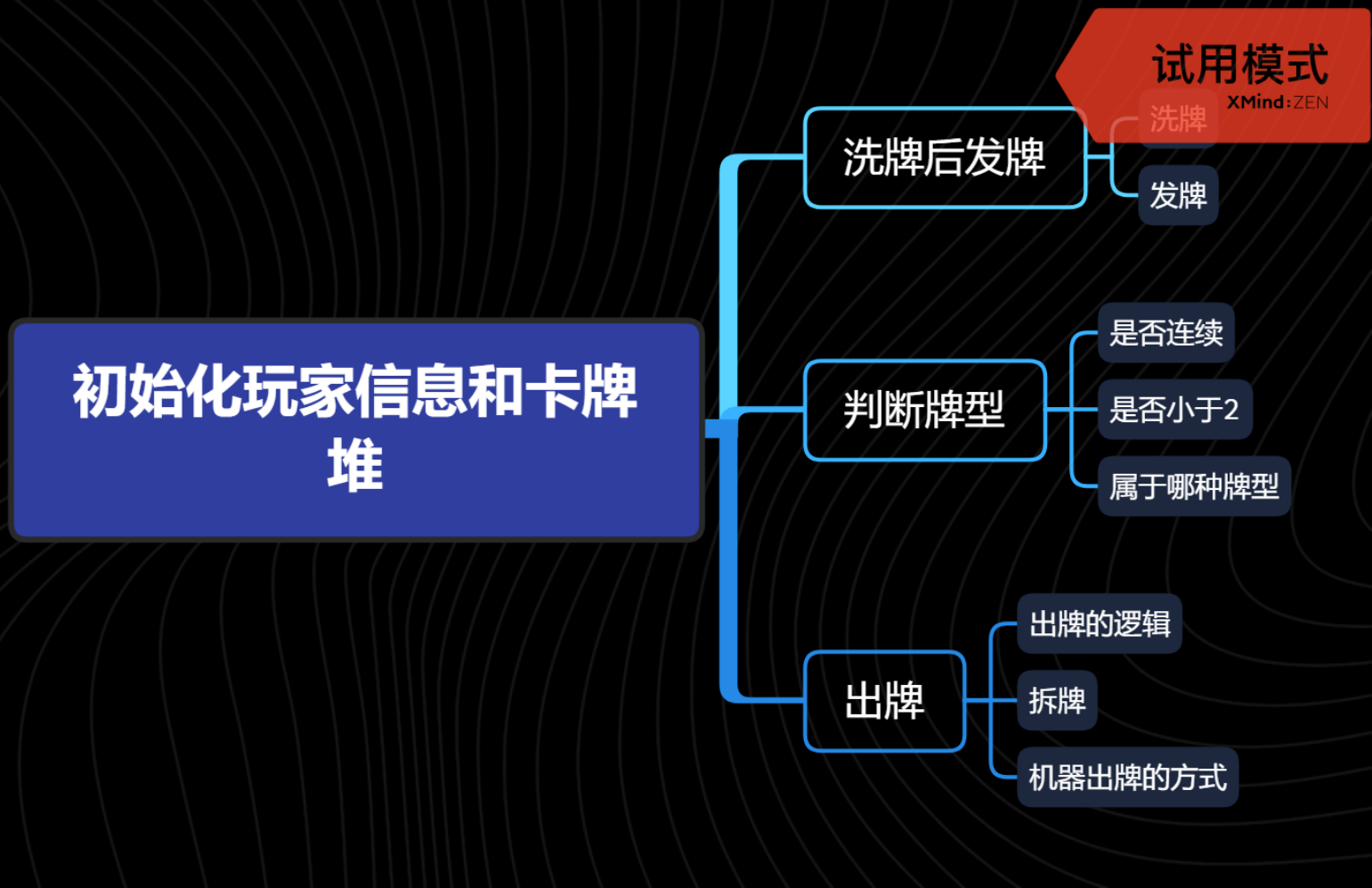
发牌

```cpp
//---GameScene.cpp  //代码有节略
void GameScene::DealCards() {
    index_fapai = 0;
    srand(time(0));
    std::random_shuffle(_pokeInfo.begin(), _pokeInfo.end());// 洗牌
}

void GameScene::DealCardsCallback(Node* node){
    if (index_fapai < 51) {
        if (index_fapai % 3 == 0)  // 角色1
            _player1->DealCards(this, _pokeInfo.at(index_fapai));
        else if (index_fapai % 3 == 1)// 角色2
            _player2->DealCards(this, _pokeInfo.at(index_fapai));
        else if (index_fapai % 3 == 2) // 角色3
            _player3->DealCards(this, _pokeInfo.at(index_fapai));
        index_fapai++;
    }
    else
        _menuQiangDiZhu->setVisible(true);
}

void GameScene::menuQiangCallback(Ref* pSender){
    DealLandlordCards(_player1);    // 分发底牌
}

void GameScene::menuBuQiangCallback(Ref* pSender)
{
    auto callback = CallFuncN::create(CC_CALLBACK_1(GameScene::Qiang2Callback, this));
}

void GameScene::Qiang2Callback(Node* node)
{
    if (_player2->IsQiangLandlord())// 如果玩家不抢地主，则由下家随机
    {
      DealLandlordCards(_player2);
      auto callback = CallFuncN::create(CC_CALLBACK_1(GameScene::ChuPai2Callback, this));
    }
    else{
        auto callback = CallFuncN::create(CC_CALLBACK_1(GameScene::Qiang3Callback, this));
    }
}
//---Player.cpp
void Player::DealCards(GameScene* scene, PokerInfo info)
{
    auto card = Poker::create(info, !_isHero);
    _cardsManager->sortAllChildren();
    updateCards();
}
```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

```cpp
enum PokerNum {//扑克牌值,依次增大 ---Poker.h
    NUM_3 = 0,
    NUM_4,
    NUM_5,
    NUM_6,
    NUM_7,
    NUM_8,
    NUM_9,
    NUM_10,
    NUM_J,
    NUM_Q,
    NUM_K,
    NUM_A,
    NUM_2,
    NUM_XW,//小王
    NUM_DW//大王
};

//Common.cpp
bool IsContinuous(std::vector<int>& vecPoker)
{
    bool ret = true;// 排序
    std::sort(vecPoker.begin(), vecPoker.end());
    // 排序完成后比较相邻的两个数字的差值，如果全为1，则为连续 for (int i = 0; i < vecPoker.size() - 1; i++){
        if (vecPoker[i + 1] - vecPoker[i] != 1){
            ret = false;
            break;
        }
    }
    return ret;
}

bool IsLess2(std::vector<int>& vecPoker)
{
    bool ret = false;
    for (int i = 0; i < vecPoker.size(); i++)
    {
        ret = vecPoker[i] >= 12;
    }

    return !ret;
}
```



判断牌型

```
CARDS_DATA JudgePaiXing(std::vector<int>& cards)
{
    CARDS_DATA ret;
    unsigned int length = 牌的数量
    std::sort(cards.begin(), cards.end());

    for (int i = 0; i < length; i++){
        ret._cards.push_back(cards[i]);
    }

    //小于5张牌
    if 牌数大于0且小于5
        if cards的第一张牌和最后一张牌相同
            if length == 4
                炸弹
            else
                另外三种牌
            return ret;
        // 火箭
        if 牌数为2,且第一张是小王,第二张是大王
            return ret;
        // 三带一
        if 牌数为4,且:第1张与第3张相同 或 第2张与第4张相同
            比如排序后999J 或3999
            if 第1张与第3张相同
                ret._value = 40 + cards[0];其大小取决于第1张牌
            else 第2张与第4张相同
                ret._value = 40 + cards[1];其大小取决于第2张牌
            return ret;
        }
    endif

    else if (length >= 5)// 大于等于5张牌
        // 连牌
        if 如果此牌连续,并且都小于2(因为2不能一起连)
            ret._value = 60 + cards[0];其大小取决于第1张牌
            return ret;
        // 连对
        if 大于六张，且为牌数双数
            bool is_all_double = true;// 判断是否都是对子
            for (int i = 0; i < length; i += 2){   以2为跨度
                if (cards[i] != cards[i + 1]){     每次遍历的这张牌与下一张牌必须相同
                    is_all_double = false;
                    break;
                }
            }
            // 判断对子是否连续
            if (is_all_double){
                std::vector<int> vec_single;
                for (int i = 0; i < length; i += 2)以2为跨度
                    vec_single.push_back(cards[i]);
                if (IsContinuous(vec_single))每次遍历的这张牌与下一张对子的起始牌必须
连续            {
                    ret._value = 80 + cards[0];其大小取决于第一张牌
                    return ret;
                }
            }
        }


```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

判断牌型2

```
备注:
struct CRAD_INDEX
{
    std::vector<int>    single_index;   //记录单张的牌
    std::vector<int>    double_index;   //记录对子的牌
    std::vector<int>    three_index;        //记录3张
    std::vector<int>    four_index;     //记录4张
};
上一张图片中的函数续:
        CRAD_INDEX card_index;
        for (int i = 0; i < length; ){
            if 不越界且 cards[i] == cards[i + 1]
                if  不越界且 && cards[i + 1] == cards[i + 2]
                    if  不越界且 && cards[i + 2] == cards[i + 3] 记录4张的牌组合-炸弹 i +=
4;                  else 记录3张组合, i += 3;
                else 记录对子组合, i += 2;
            else 记录单牌, i++;
        }
        // 3带2
        if  3张相同牌的牌组 和 对子数量大于1 且没有炸弹和单牌
            {ret._type = THREE_TWO_CARD;return ret;}
        // 飞机    前提：两个连续三张的
        if  如果3张相同的牌组 数量大于2,且数字连续(相差1)
            if 没有单牌且没有对子// 333444
                {ret._type = AIRCRAFT_CARD;return ret;}
            if 2张单牌且没有对子// 33344456
                {ret._type = AIRCRAFT_SINGLE_CARD;return ret;}
            if 没有单牌且有1个对子// 33344455 相当于把1个对子拆成2张单牌
                {ret._type = AIRCRAFT_SINGLE_CARD;return ret;}
            if 没有单牌且有2个对子// 3334445566
                {ret._type = AIRCRAFT_DOUBLE_CARD;return ret;}
        // 4带2
        if 有四张相同的牌,且没有连续的三张牌
            if 有2张单牌,且没有对子// 444423
                {ret._type = BOMB_TWO_CARD;return ret;}
            if 没有单牌且有1个对子// 444422
                {ret._type = BOMB_TWO_CARD;return ret;}
            if 没有单牌且有2个对子// 44442233
            {ret._type = BOMB_TWOOO_CARD;return ret;}

    ret._type = ERROR_CARD;
    ret._value = 0;
    return ret;
}
```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

出牌逻辑

```
void GameScene::menuChuPaiCallback(Ref* pSender) 对于每个玩家,修改player123的顺序即可
{
    auto player3出的牌 = 获取_player3刚刚出的牌;
    CARDS_DATA player3_card_data = 判断牌型(player3出的牌); // 出牌之前，判断上家的牌型

    if (player3没出牌)
    {
        auto player2_outcards = 获取_player2刚刚出的牌;
        CARDS_DATA player2_card_data = 判断牌型(player2出的牌);
        if (player2没出牌) _player1出牌
        else  _player1跟牌
    }
    else _player1跟牌
}

void GameScene::OutCard(float delta)
{
    if (_state == 1){// 开局
      auto player3出的牌 = 获取_player3刚刚出的牌;
      CARDS_DATA player3_card_data = 判断牌型(player3出的牌); // 出牌之前，判断上家的牌
型
            if (player3没出牌)
                auto player2_outcards = 获取_player2刚刚出的牌;
                CARDS_DATA player2_card_data = 判断牌型(player2出的牌);
                if (player2没出牌) _player1出牌
                else// 提示部分未完善
                    if (牌型相同且大于player2出的牌 或者 出炸弹或火箭且大于player2的炸弹
                        _player1出牌
                    else _player1不出牌
            else if 牌型相同且大于player3出的牌 或者 出炸弹或火箭且大于player3的炸弹
                        _player1出牌
                    else _player1不出牌
 }
```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

跟牌

```cpp
// 牌型
enum  CARD_TYPE{
    SINGLE_CARD = 1,            //单牌
    DOUBLE_CARD,            //对子
    THREE_CARD,             //3不带
    BOMB_CARD,              //炸弹
    MISSILE_CARD,           //火箭
    THREE_ONE_CARD,         //3带1
    THREE_TWO_CARD,         //3带2
    BOMB_TWO_CARD,          //四个带2张单牌
    BOMB_TWOOO_CARD,        //四个带2对
    CONNECT_CARD,           //连牌
    COMPANY_CARD,           //连对
    AIRCRAFT_CARD,          //飞机不带
    AIRCRAFT_SINGLE_CARD,   //飞机带单牌
    AIRCRAFT_DOUBLE_CARD,   //飞机带对子
    ERROR_CARD              //错误的牌型
};
std::vector<int>& Player::FindFollowCards(CARD_TYPE cardType, unsigned int count, unsigned int value)  跟牌
{
    for (int i = 0; i < _allCardGroups.size(); i++){
    if (_allCardGroups[i]._value <= value)// 先判断牌值，如果牌值小直接下一轮
            continue;
        // 单张/对子/三不带/炸弹/单顺/双顺/飞机/火箭
        if 当前牌型和需要跟的牌型相同
            if (cardType == CONNECT_CARD || cardType == COMPANY_CARD || cardType == AIRCRAFT_CARD)
                if 当前牌型所需的数量与手牌的数量一样(否则没有必要拆开)
                    return 当前遍历到的牌组
            else return 当前遍历到的牌组
        else// 三带一/三带一对
            if 上家出牌为3带1
                if 当前遍历的到的为3张相同牌
                    for (int j = 0; j < _allCardGroups.size(); j++)
                        if 当前遍历的到的为1张单牌
                            将两组牌放入_vecFindFollowCards
                            return _vecFindFollowCards;
            else if 上家出牌为3带一对
                if (_allCardGroups[i]._type == THREE_CARD)
                    for (int j = 0; j < _allCardGroups.size(); j++)
                        if (_allCardGroups[j]._type == DOUBLE_CARD && _allCardGroups[i]._cards[0] != _allCardGroups[j]._cards[0])
                            将牌放入_vecFindFollowCards
                            return _vecFindFollowCards;

    for (int i = 0; i < _allCardGroups.size(); i++)// 没找到对应的牌型并且牌值大于上家的，使用炸弹
        if  判断牌值，如果牌值小则跳过 continue;
        if 找到炸弹 return 当前遍历到的炸弹;

    for (int i = 0; i < _allCardGroups.size(); i++) // 没找到对应的牌型并且牌值大于上家的，使用火箭（王炸）
    {
        if 找到火箭(王炸) return 当前遍历到的火箭
    }
    return _vecFindFollowCards;
}

```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

出牌方式

```cpp
std::vector<int>& Player::FindOutCards()
{
    _vecFindFollowCards.clear();// 不同牌组优先级: 双顺 > 单顺 > 三带 > 对子 > 单牌 > 炸弹 > 火箭
    //双顺
    int index_company_card = -1;
    int count_company_card = 0;
    for (int i = 0; i < _allCardGroups.size(); i++)
    {
        if 找到双顺(连对) 并获得尽可能多的对子
            count_company_card = _allCardGroups[i]._cards.size();
            index_company_card = i;
    }
    if (index_company_card != -1)
        return _allCardGroups[index_company_card]._cards;
    //单顺
    int index_connect_card = -1;
    int count_connect_card = 0;
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 找到单顺,并继续寻找是否还有接续的牌
            count_connect_card = _allCardGroups[i]._cards.size();
            index_connect_card = i;
    if (index_connect_card != -1)
    {
        return _allCardGroups[index_connect_card]._cards;
    }
    //三不带/三带一/三带一对
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 找到三张相同的牌
            for (int j = 0; j < _allCardGroups.size(); j++)
                if 找到一张单牌
                  将牌放入_vecFindFollowCards
                    return _vecFindFollowCards;
                return _allCardGroups[i]._cards;
    //对子
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 遍历到对子 return _allCardGroups[i]._cards;
    //单牌
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 遍历到单牌 return _allCardGroups[i]._cards;
    //炸弹
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 遍历到炸弹 return _allCardGroups[i]._cards;
    //火箭
    for (int i = 0; i < _allCardGroups.size(); i++)
        if 遍历到火箭 return _allCardGroups[i]._cards;
    std::vector<int> tmp;
    return tmp;
}
```

初始化玩家信息和卡牌堆

- 洗牌后发牌
  - 洗牌
  - 发牌
- 判断牌型
  - 是否连续
  - 是否小于2
  - 属于哪种牌型
- 出牌
  - 出牌的逻辑
  - 拆牌
  - 机器出牌的方式

```cpp
void GameScene::GoToGameOverScene(Ref* pSender)
{
    auto scene = GameOver::createScene();
    Director::getInstance()->replaceScene(scene);
}

Scene* GameOver::createScene()
{
    auto layer = GameOver::create();
    auto scene = Scene::create();
    scene->addChild(layer);
    return scene;
}
bool GameOver::init()
{
    if (!Layer::init())
    {
        return false;
    }

    auto visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    auto OverTitle = MenuItemImage::create("GameOverPic.png", "GameOverPic.png");
    auto mainmenuItem = MenuItemFont::create("Return to Main
Menu",CC_CALLBACK_1(GameOver::GoToMainMenuScene,this));
    auto endItem = MenuItemFont::create("Exit", CC_CALLBACK_1(GameOver::GoExitGame, this));
    auto menu = Menu::create(OverTitle, mainmenuItem, endItem,NULL);
    menu->alignItemsVerticallyWithPadding(visibleSize.height / 8);
    this->addChild(menu);
    return true;
}

```

4.4 场景类（Scene）

Scene 是场景类，它相当于一个大容器，将包含在内的层和精灵输出到屏幕上，是整个树的根节点。其实 Scene 的内部构成非常简单，虽然继承自 Node，但没有在 Node 的基础上增加任何成员变量和方法，只是重构了 init。由此可以看出，Scene 并没有屏显的作用，

其作用只是承上启下。节点只有被加到树中才会更新逻辑及绘制，绘制的方法 visit 是节点实现的，场景只是把节点添加到树中使其可以执行该函数，然后导演类激活场景实例，使它构成的树生效（树可以有多个，但只有导演类激活的树才有效，在 Cocos2d 中导演类最多只能激活一个树）。它用 setContentSize 方法将屏幕的尺寸传递给场景，使其默认和屏幕一样大，将锚点设置为(0.5,0.5)并将其锁定。

```cpp
void GameScene::GoToGameOverScene(Ref* pSender)
{
    auto scene = GameOver::createScene();
    Director::getInstance()->replaceScene(scene);
}

Scene* GameOver::createScene()
{
    auto layer = GameOver::create();
    auto scene = Scene::create();
    scene->addChild(layer);
    return scene;
}
bool GameOver::init()
{
    if (!Layer::init())
    {
        return false;
    }

    auto visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    auto OverTitle = MenuItemImage::create("GameOverPic.png", "GameOverPic.png");
    auto mainmenuItem = MenuItemFont::create("Return to Main
Menu",CC_CALLBACK_1(GameOver::GoToMainMenuScene,this));
    auto endItem = MenuItemFont::create("Exit", CC_CALLBACK_1(GameOver::GoExitGame, this));
    auto menu = Menu::create(OverTitle, mainmenuItem, endItem,NULL);
    menu->alignItemsVerticallyWithPadding(visibleSize.height / 8);
    this->addChild(menu);
    return true;
}

```

2020/6/7

4.5 布景层类（Layer）

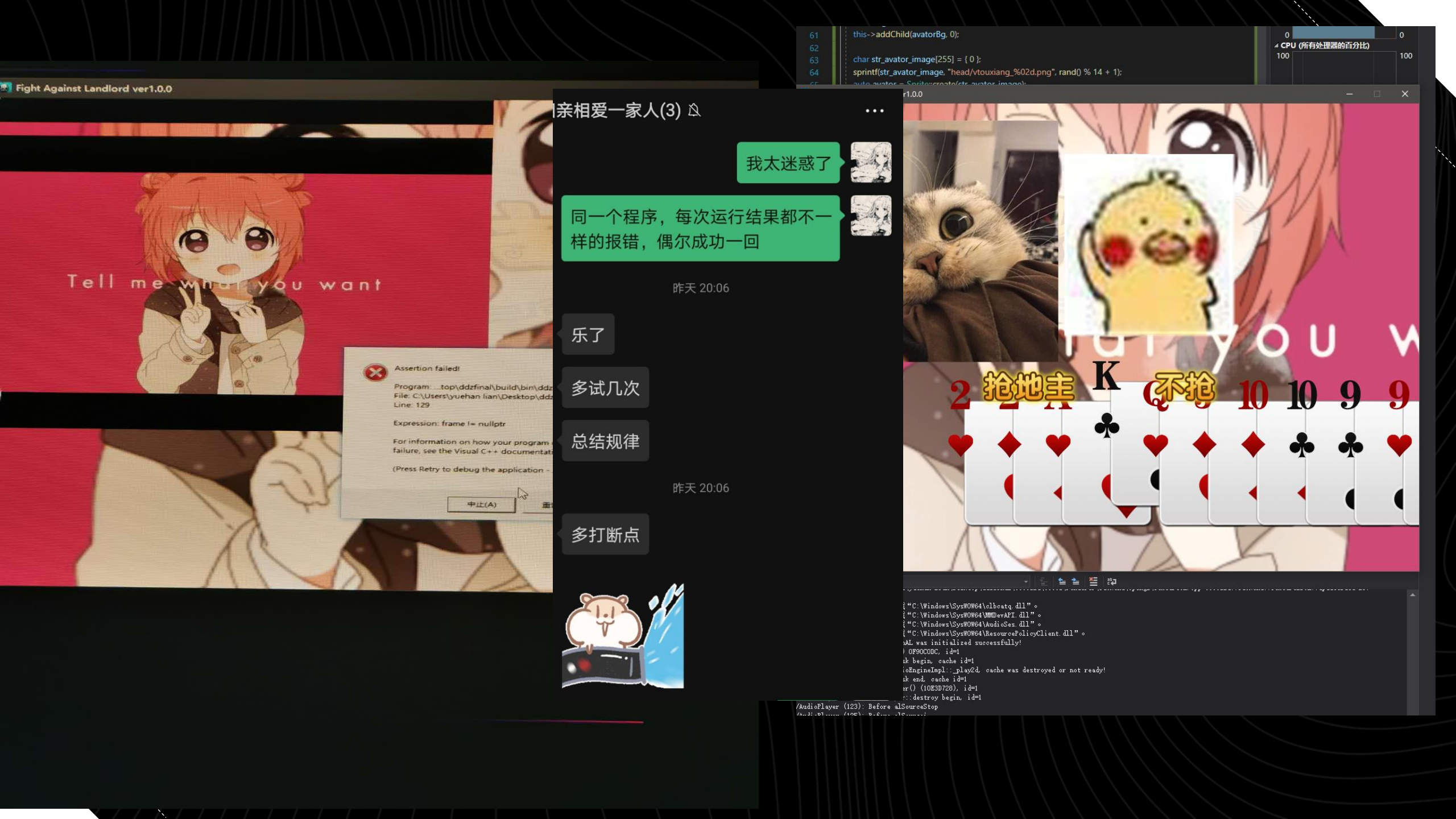
Layer 是在游戏开发中常用的类，通常将其添加到场景中，然后再将其他精灵添加到该 Layer 上。Layer 的继承关系如图 4-7 所示。

在一个游戏中可以有很多场景，每个场景里面又可能包含多个图层，这里的图层一般就是 Layer 对象。Layer 本身几乎没什么功能，相对于 Node，Layer 可用于接收触摸和加速计输入。其实，Cocos2d 对图层并没有严格的要求，图层不一定要使用 Layer 类，它也可以是一个简单的 Node，为什么呢？因为我们新建一个图层是为了能够容纳更多的子节点，Node 也可以添加子节点，所以，如果你的图层不需要接收触摸和加速计输入，就尽量使用 Node 表示图层，Layer 能够接收触摸和加速计输入，会增加不必要的开销。移动、缩放、旋转整个图层，图层上的所有节点也会跟着移动、缩放、旋转。由图 4-7 可以看出，Layer 类继承自 Node 类，并且 Layer 类还遵循触屏代理协议、加速度传感器代理协议、键盘时间代理协议等。除此之外，Layer 类还有子类，我们列举一下常用的子类，如图 4-8 所示。

```cpp
bool Poker::onTouchBegan(Touch* touch, Event* event)
{
    if (getRect().containsPoint(convertTouchToNodeSpaceAR(touch)))
    {
        click();
        return true;
    }
    return false;
}

void Poker::onTouchEnded(Touch* touch, Event* event)
{
}

void Poker::onTouchCancelled(Touch* touch, Event* event)
{
}

void Poker::onTouchMoved(Touch* touch,Event* event)
{
}
```

（1）首先在创建层时设置开启触摸。

```cpp
this->_touchEnabled = true;
//获取事件分发器
auto dispatcher = Director::getInstance()->getEventDispatcher();
//设置单点触摸
auto touchListener = EventListenerTouchOneByOne::create();
//设置多点触摸
auto touchListener = EventListenerTouchAllAtOnce::create();
//设置是否向下传递触摸
touchListener ->setSwallowTouches(true);
//设置监听器的具体处理函数
touchListener->onTouchBegan = CC_CALLBACK_2(HelloWorld:: onTouchBegan,this);
touchListener->onTouchMoved = CC_CALLBACK_2(HelloWorld:: onTouchMoved,this);
touchListener->onTouchEnded = CC_CALLBACK_2(HelloWorld:: onTouchEnded,
this);
touchListener-> onTouchCancelled = CC_CALLBACK_2(HelloWorld:: onTouchCancelled,
this);
```


Fight Against Landlord ver1.0.0
Tell me what you want
Assertion failed!
Program: ...top\ddzfinal\build\bin\ddz
File: C:\Users\yuehan lian\Desktop\ddz
Line: 129
Expression: frame != nullptr
For information on how your program
failure, see the Visual C++ documentati
(Press Retry to debug the application
中止(A)
亲相爱一家人(3)
我太迷惑了
同一个程序，每次运行结果都不一样的报错，偶尔成功一回
昨天 20:06
乐了
多试几次
总结规律
昨天 20:06
多打断点
this->addChild(avatorBg, 0);
char str_avator_image[255] = { 0 };
sprintf(str_avator_image, "head/vtouxiang_%02d.png", rand() % 14 + 1);
CPU (所有处理器的百分比)
抢地主
不抢
2
K
Q
10
9
"C:\Windows\SysWOW64\clbcatq.dll"
"C:\Windows\SysWOW64\MMDevAPI.dll"
"C:\Windows\SysWOW64\AudioSes.dll"
"C:\Windows\SysWOW64\ResourcePolicyClient.dll"
AL was initialized successfully!
OF90CODC, id=1
k begin, cache id=1
oEngineImpl::_play2d, cache was destroyed or not ready!
k end, cache id=1
r() (10E3D728), id=1
::destroy begin, id=1
/AudioPlayer (123): Before alSourceStop


谢 谢