同濟大學
TONGJI UNIVERSITY

Completion and Review of Teamfight Tactics Project

2250758 林继申 (组长)

2251730 刘淑仪

2252712 杨兆镇

2252843 杨宇琨

GitHub: https://github.com/MinmusLin/Teamfight_Tactics

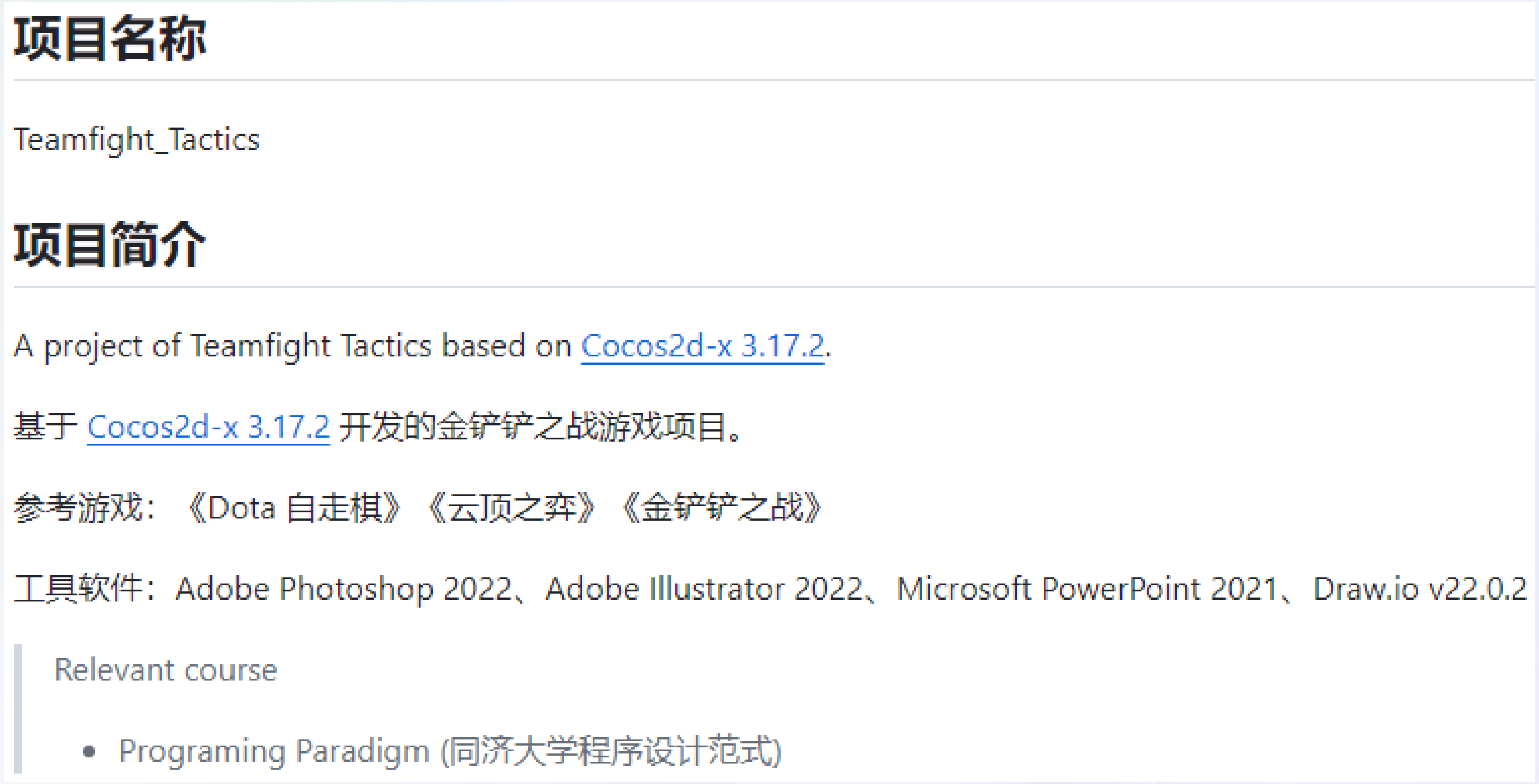
Speaker: 杨宇琨 林继申

2024 - 01 - 02

Brief Introduction

Introduction to team roles and responsibilities, technology stack and tools, schedule and timeline, functional implementation and bonus points

Brief Introduction



项目名称

Teamfight_Tactics

项目简介

A project of Teamfight Tactics based on Cocos2d-x 3.17.2.

基于 Cocos2d-x 3.17.2 开发的金铲铲之战游戏项目。

参考游戏：《Dota 自走棋》《云顶之弈》《金铲铲之战》

工具软件：Adobe Photoshop 2022、Adobe Illustrator 2022、Microsoft PowerPoint 2021、Draw.io v22.0.2

> Relevant course
>
> - Programing Paradigm (同济大学程序设计范式)

Team Roles and Responsibilities

林继申

项目后端工作统筹
项目任务分工
代码审查
项目框架搭建
网络环境搭建
练习模式与联机模式
`AppDelegate` 类
`Battle` 类
`HoverButton` 类
`Champion` 类
`Control` 类及其派生类
`ChampionAttributesLayer` 类
`PlacementMarkerLayer` 类
`ScoreBoardLayer` 类
`LocationMap` 类
`Player` 类及其派生类
`Scene` 类及其派生类
`Server` 类

刘淑仪

项目前端工作统筹
练习模式与联机模式
`Champion` 类
`ChampionAttributesLayer` 类
`PlacementMarkerLayer` 类
`ScoreBoardLayer` 类
`Scene` 类及其派生类
图标绘制
按钮绘制
战斗英雄与小小英雄绘制
图像元素绘制
场景绘制
音频引擎
背景音乐与音效

杨宇琨

项目测试统筹
练习模式与联机模式
`Battle` 类
`Champion` 类
`OfflineModeControl` 类
`HumanPlayer` 类
`OfflineModeBattleScene` 类
`OnlineModeBattleScene` 类
战斗英雄绘制

杨兆镇

项目 AI 玩家算法统筹
练习模式
`Player` 类及其派生的 `AIPlayer` 类

Contributions (git log)

姓名	学号	代码行数	工作量
林继申 (组长)	2250758	5082	35%
刘淑仪	2251730	1385	25%
杨兆镇	2252712	467	15%
杨宇琨	2252843	2282	25%

Technology Stack and Tools

Cocos2d-x 3.17.2

Compared to cocos2d-x 4.0, there are richer documentation and tutorial resources available, and the development environment is adequate.

C++

When developing games within the Cocos2d framework using the C++ language, it provides performance advantages and flexibility.

Git

Git effectively supports team collaboration, aiding teams in working together more efficiently during project development by facilitating communication and code management.

TONGJI UNIVERSITY

Schedule and Timeline

2023.11.22~2023.12.4

Stage One:
Setting up the environment to learn Cocos2d-x
Understanding code conventions, learning Markdown
Creating a repository, mastering Git usage.

2023.12.8~2023.12.12

Stage Two:
Configure socket networking environment.
Set up the basic framework. Implement the AppDelegate class and the Server class to lay the foundation for multiplayer mode.

2023.12.16~2023.12.22

Stage Three:
Implemented various functionalities for program-user interaction. Completed drawing related hero and scene images. Preliminarily implemented the necessary classes for the combat scene.

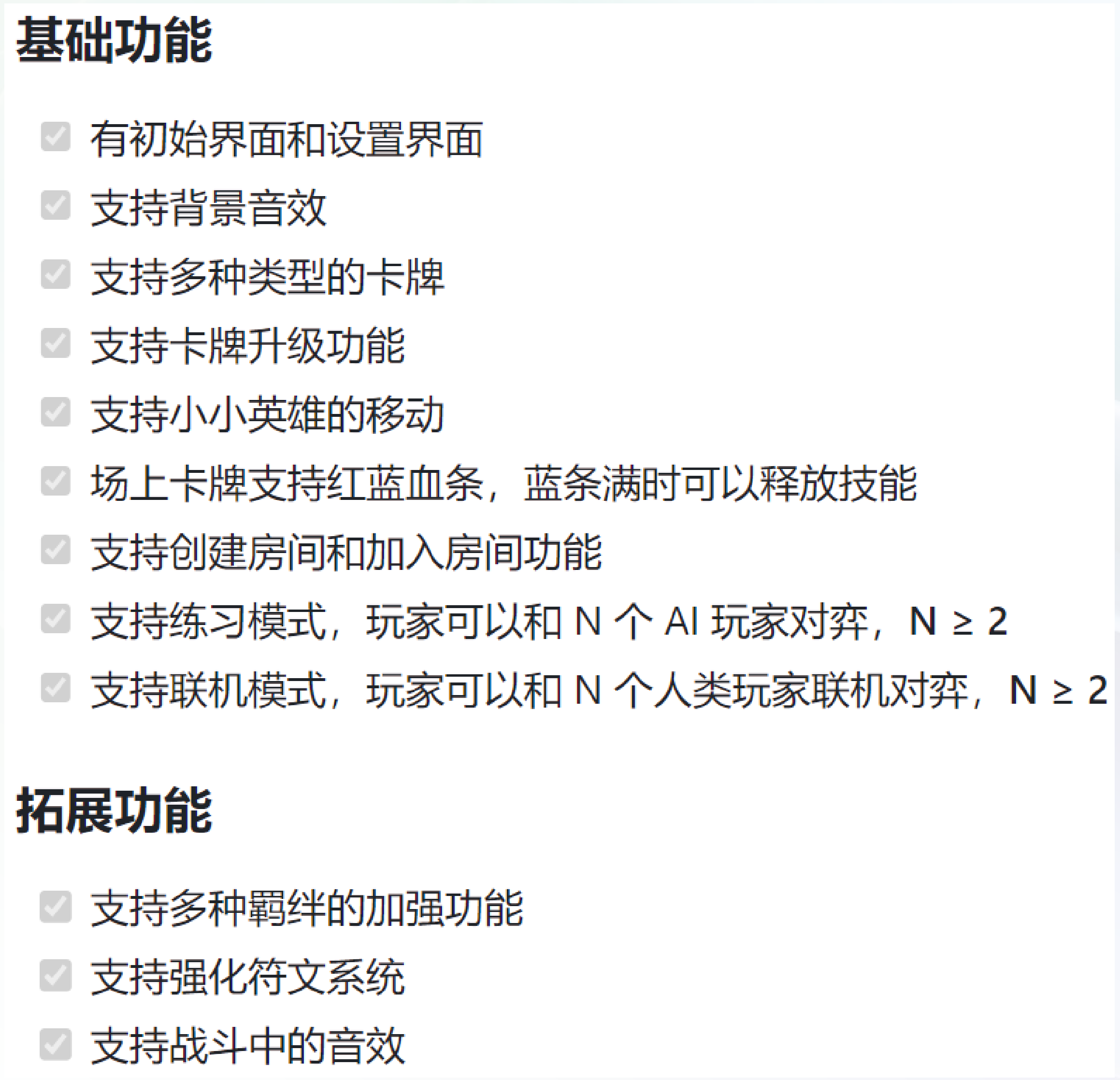
2023.12.23~2023.12.27

Stage Four:
Implemented the battle scene and AI algorithm.
Enhanced the overall game logic by integrating all basic functionalities.

2023.12.28~2023.12.31

Stage Five:
Implemented the battle scene and AI algorithm.
Enhanced the overall game logic by integrating all basic functionalities.

Functional Implementation



基础功能

- ✓ 有初始界面和设置界面
- ✓ 支持背景音效
- ✓ 支持多种类型的卡牌
- ✓ 支持卡牌升级功能
- ✓ 支持小小英雄的移动
- ✓ 场上卡牌支持红蓝血条，蓝条满时可以释放技能
- ✓ 支持创建房间和加入房间功能
- ✓ 支持练习模式，玩家可以和 N 个 AI 玩家对弈，N ≥ 2
- ✓ 支持联机模式，玩家可以和 N 个人类玩家联机对弈，N ≥ 2

拓展功能

- ✓ 支持多种羁绊的加强功能
- ✓ 支持强化符文系统
- ✓ 支持战斗中的音效

Bonus Points

- **版本控制和团队协作**

✓ 合理使用 Git 控制版本，将项目开源至 Github

✓ 团队成员分工合理平等

- **代码质量和安全**

✓ 使用单元测试保证代码质量

✓ 合理地抛出异常和处理

- **功能和架构**

✓ 界面精美

✓ 项目目录结构清晰

- **其他加分项**

✓ 没有内存泄露

✓ 程序运行过程不会发生崩溃情况

✓ 尽可能多地使用了 C++11 特性

类型推导、构造函数参数列表初始化、基于范围的 for 循环、继承构造函数、空指针关键字 nullptr、修饰常量 constexpr、新随机数、lambda 表达式、时间库 chrono 和线程库 this_thread、Unicode 编码支持

✓ 商店推荐系统与 AI 落棋算法

✓ 规范统一的代码风格

✓ 游戏还原度高

✓ 常变量的集中定义

同濟大學 TONGJI UNIVERSITY

General Design

An outline of out project

Design Philosophy and Principles

High Cohesion, Low Coupling

01 → 02 → 03

Definition of Game Mechanics and Game States

Define the core mechanics, objectives, and rules of the game. Define the various states the game may be in (such as main menu, gameplay, game over). Plan and design a single game scene.

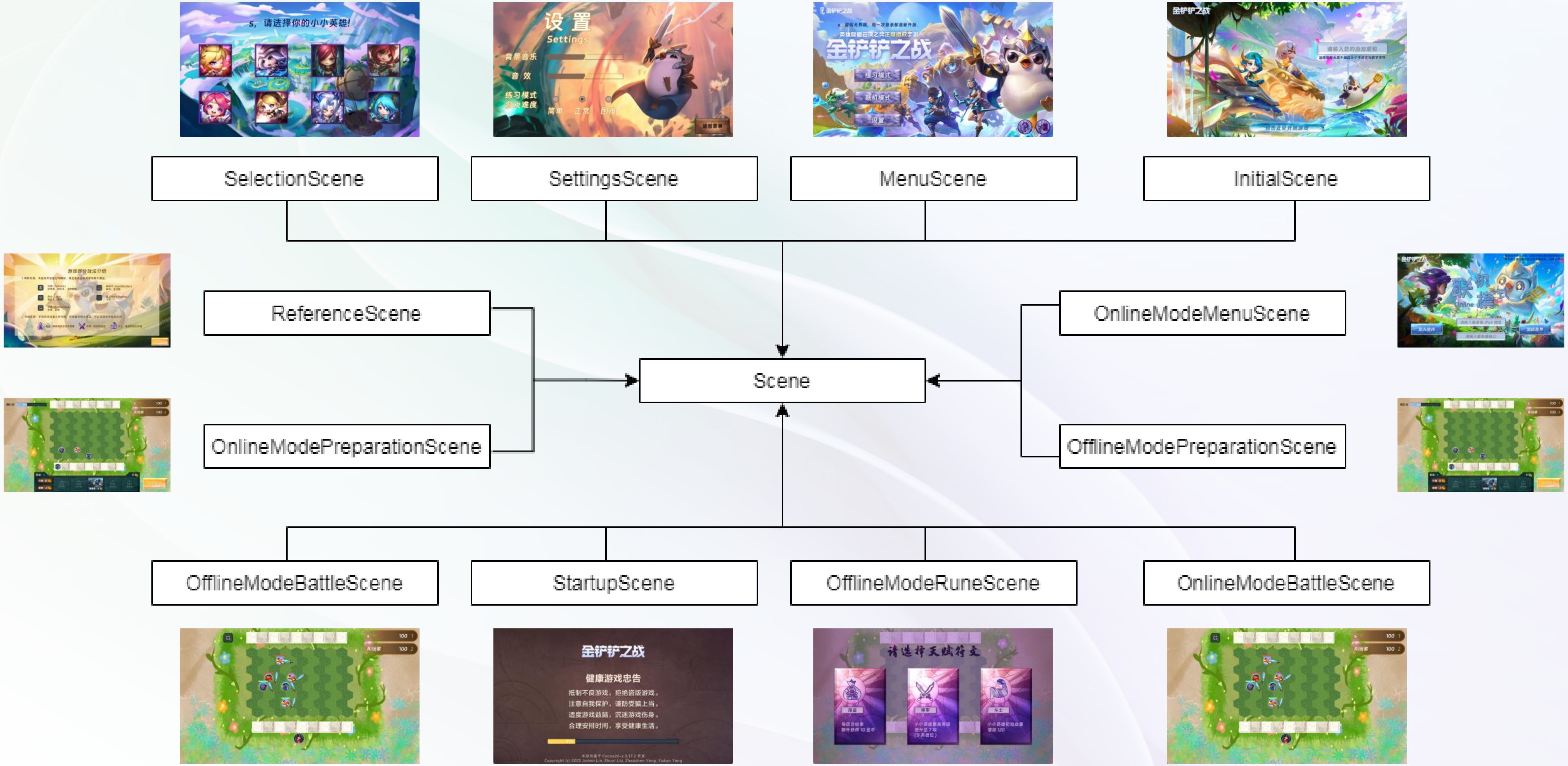
Class Planning and File Classification

Based on the planned game flow, organize the classes for the objects appearing in the game (primarily categorized into base classes, derived classes, method classes, and management classes), implementing them in different files.

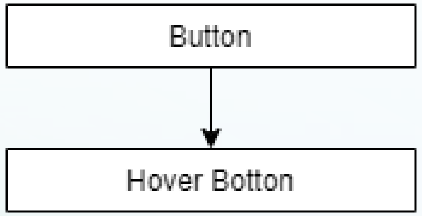
Refactoring

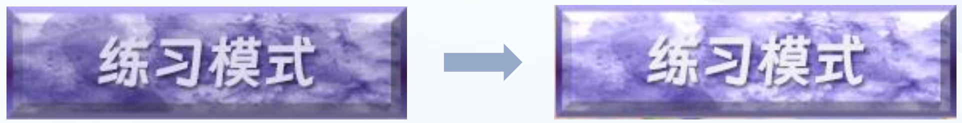
This aims to enhance code readability, maintainability, making it easier to understand, modify, and extend. Refactoring does not involve adding new features but focuses on improving the internal structure of the codebase, eliminating redundancy, and enhancing its overall quality.

Class Scene


SelectionScene
SettingsScene
MenuScene
InitialScene
ReferenceScene
OnlineModeMenuScene
Scene
OnlineModePreparationScene
OfflineModePreparationScene
OfflineModeBattleScene
StartupScene
OfflineModeRuneScene
OnlineModeBattleScene

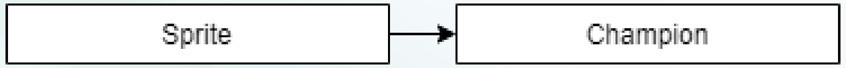
Class HoverBotton



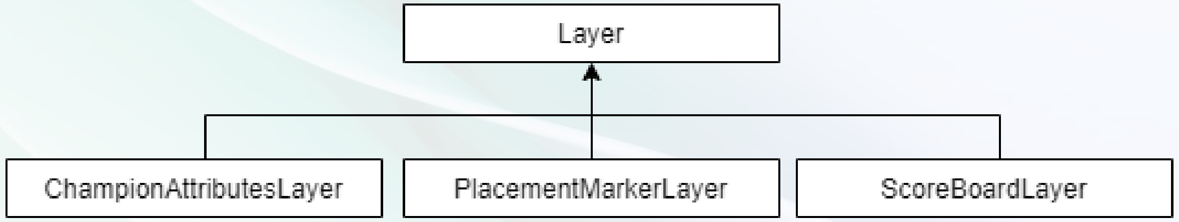


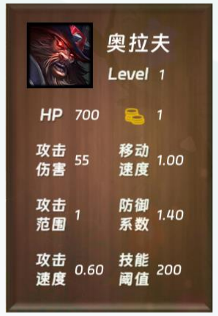
(Hover)

Class Champion

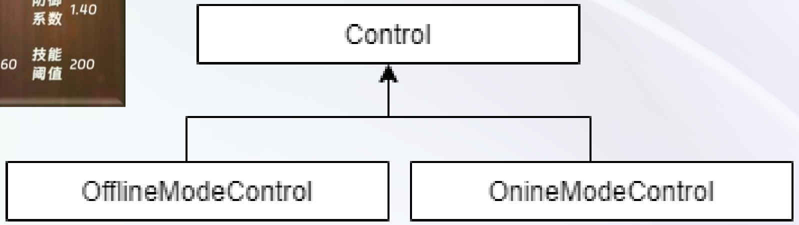


Class Layer

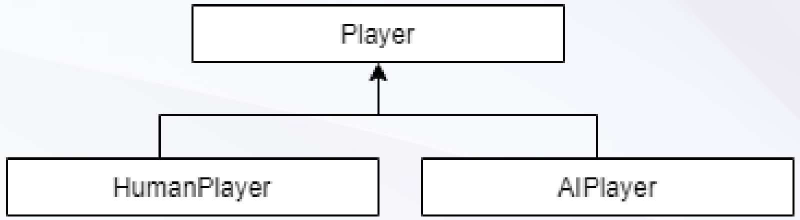




Class Control

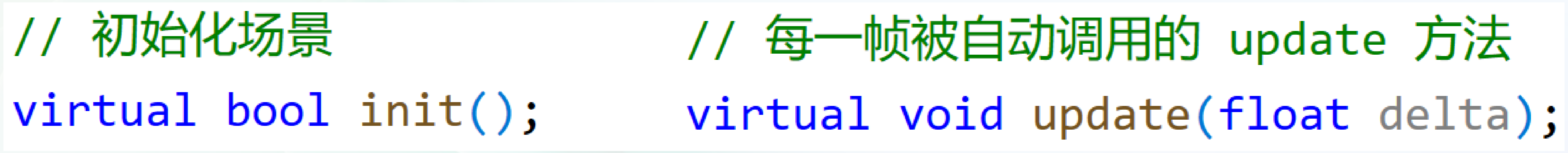


Class Player



Manager Class

Polymorphism



```
// 初始化场景                        // 每一帧被自动调用的 update 方法
virtual bool init();                 virtual void update(float delta);
```

In this code snippet, a virtual method is declared. This method's declaration uses the virtual keyword, indicating that it is a virtual function. Virtual functions can be declared in the base class and overridden in derived classes. This mechanism is known as polymorphism.

Polymorphism allows different objects to respond differently to the same message.

This polymorphic behavior enables you to handle an object without needing to know its exact type, only requiring knowledge that it is of the base class type. When a virtual function is called, the version that actually runs is that of the object's class, not the base class.

Such a design facilitates the creation of more flexible, extensible code structures, allowing for extension and customization of derived classes as needed, all while using a unified interface to manage various object types.

Code Safety

The **Try-Catch** is an exception handling mechanism used to capture and manage potential exceptions or errors within a program.

When using the **Try-Catch** structure, the code block is placed within a **Try** block, which encapsulates the code that might throw an exception. If an exception occurs within the **Try** block, the program immediately jumps to the corresponding **Catch** block. This allows for the execution of specific exception handling code without causing the program to crash.

Try Block: Contains the segment of code where exceptions might be thrown.

```
try { championMap[i][j] = new Champion(championCategoryMap[i][j]); }
```

Catch Block: Used to capture and handle different types of exceptions. Multiple **Catch** blocks can be employed, with each block handling a specific type of exception.

```
catch (const std::bad_alloc& e) {
    std::cerr << "Memory allocation failed: " << e.what() << std::endl;
    throw;
}
```

Code Safety

To prevent memory leaks, we adopt a manual approach in managing dynamic memory by strictly ensuring a one-to-one match between new and delete operations. There are several instances in our project where dynamic memory deletion is meticulously managed:

1. In the Battle class, the creation of heroes is achieved through dynamic memory allocation. We create a two-dimensional pointer array championCategoryMap[BATTLE_MAP_ROWS][BATTLE_MAP_COLUMNS], where each member points to an instance of a hero on the battle map. Initially, all pointers are initialized to nullptr. Whenever a new hero is created, the respective pointer in the battle area is set to point to the hero's instance. Upon the hero's departure or at game end, we manually delete the hero variable and then update the corresponding pointer to nullptr. This practice ensures prevention against memory leaks.

2. Within the OfflineModeControl class, we create corresponding player instances based on the human player's input nickname and AI difficulty settings. We utilize dynamic memory allocation to create these objects and have a combat class representing a confirmed battle between the player and AI. To prevent memory leaks, we implement a manual deletion of corresponding objects in the destructor.

3. Similarly, within the OnlineModeControl class, we create HumanPlayer objects based on the provided player nickname parameter within the constructor. Furthermore, dynamic memory allocation is used throughout the creation of Battle objects. The creation and determination of players and battle instances significantly modularize the core functionalities of the game, ensuring systematic handling of issues.

These practices ensure that dynamic memory is allocated and released appropriately, preventing memory leaks and maintaining the program's integrity.

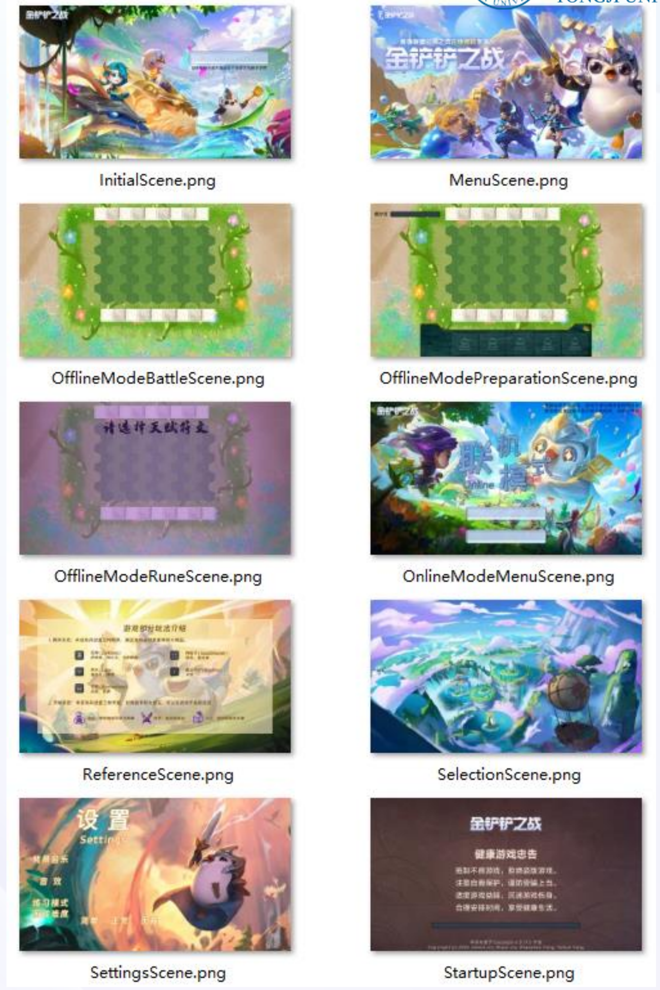
UI Design

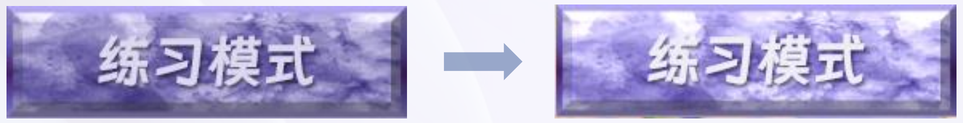
This project is designed with a total of 10 scenes based on scene transitions, namely:

1. Initial Scene (InitialScene)
2. Menu Scene (MenuScene)
3. Offline Mode Preparation / Battle Scene (OfflineModePreparation / BattleScene)
4. Offline Mode Rune Selection Scene (OfflineModeRuneScene)
5. Online Mode Menu Scene (OnlineModeMenuScene)
6. Reference Scene (ReferenceScene)
7. Selection of Mini Heroes Scene (SelectionScene)
8. Settings Scene (SettingsScene)
9. Startup Loading Scene (StartupScene)

Cocos2d-x provides two states for buttons: the normal state and the activated state. However, to ensure visually appealing graphics, this project has also designed a hover state for buttons, ensuring a responsive change when the mouse hovers over the buttons. Thus, all buttons exhibit three states: Default, Hover, and Active.

While designing the buttons, the project also took into consideration the consistency with the current scene's color scheme, aiming to create a visually harmonious and aesthetically pleasing interface.

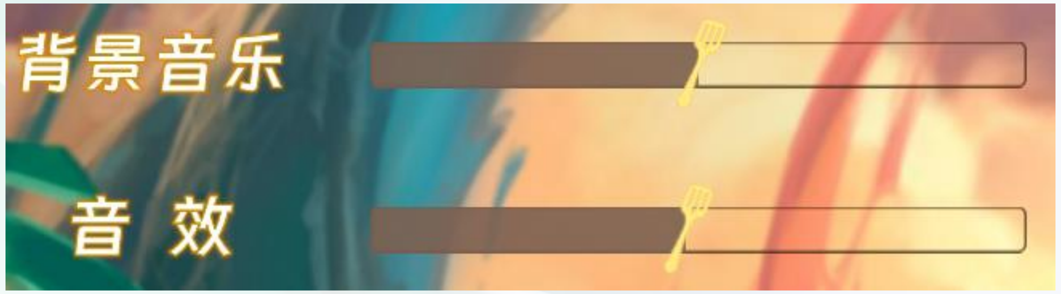




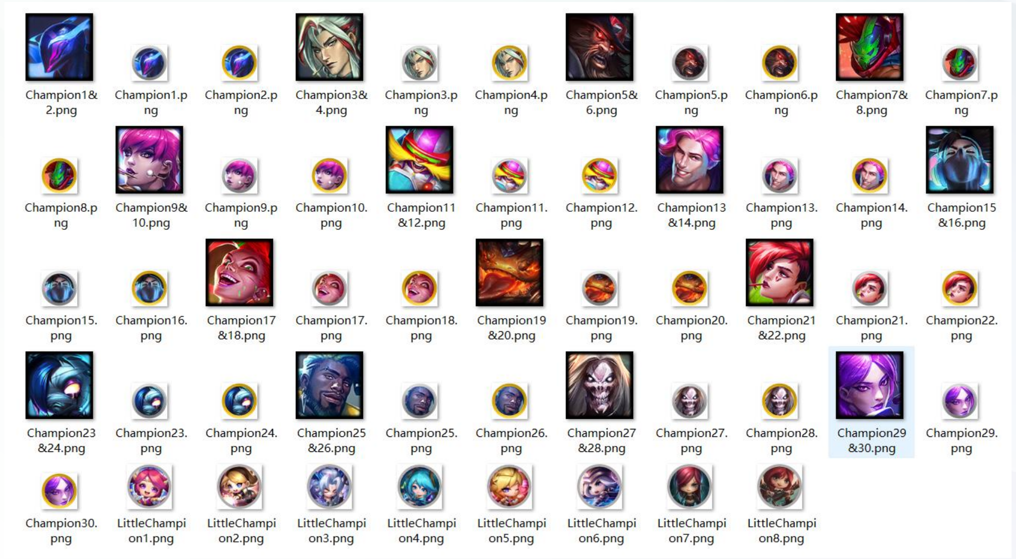
(Hover)

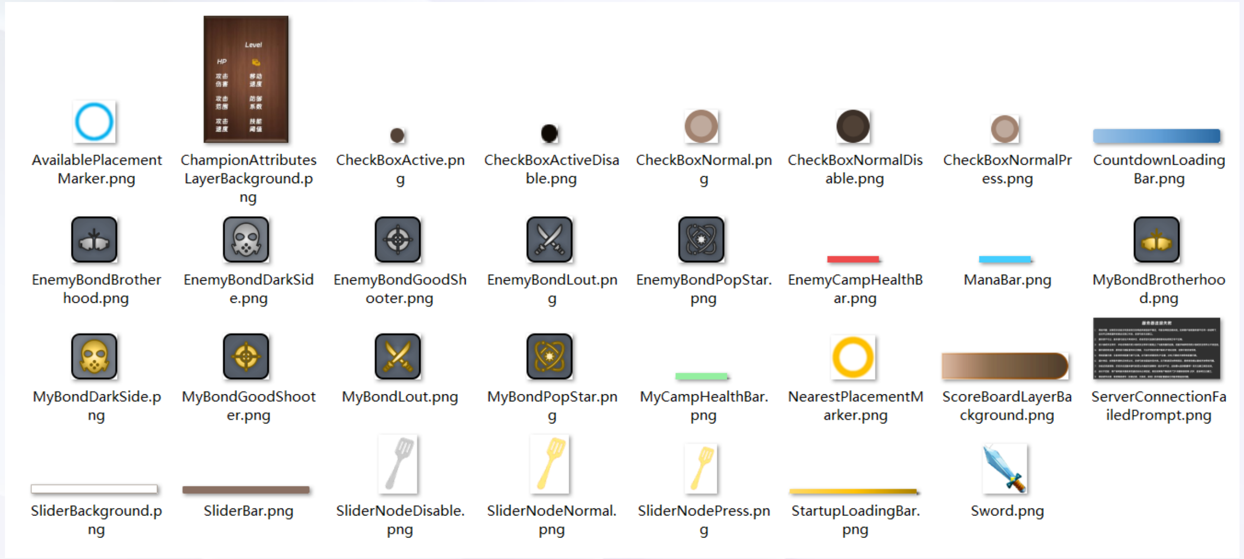
UI Design

In addition to the basic buttons, this project has also designed sliders, progress bars, and checkboxes. These elements are all designed based on the primary color scheme of the current interface. Furthermore, the slider's handle has been designed as a shovel.



The hero's attribute display labels and the store UI have been referenced from the original game, with certain modifications made to ensure harmony with the battle interface. Changes in color and opacity have been applied to maintain consistency and visual coherence with the battle interface.





Coding Style

Avoid magic numbers

```cpp
// 应用程序设置
constexpr int DESIGN_RESOLUTION_WIDTH = 1280;                                  // 设计分辨率宽度
constexpr int DESIGN_RESOLUTION_HEIGHT = 720;                                  // 设计分辨率高度
constexpr int SMALL_RESOLUTION_WIDTH = 960;                                    // 小分辨率宽度
constexpr int SMALL_RESOLUTION_HEIGHT = 540;                                   // 小分辨率高度
constexpr int MEDIUM_RESOLUTION_WIDTH = 1280;                                  // 中分辨率宽度
constexpr int MEDIUM_RESOLUTION_HEIGHT = 720;                                  // 中分辨率高度
constexpr int LARGE_RESOLUTION_WIDTH = 1920;                                   // 大分辨率宽度
constexpr int LARGE_RESOLUTION_HEIGHT = 1080;                                  // 大分辨率高度
constexpr float FRAME_RATE = 60.0f;                                            // 应用程序帧率
```

Good naming conventions

```cpp
// 练习模式游戏控制类
extern OfflineModeControl* g_offlineModeControl;

// 小小英雄种类
extern int g_littleChampionCategory;

// 创建场景
Scene* OfflineModeBattleScene::createScene()
```

File structure

```cpp
/****************************************************************
 * Project Name:  Teamfight_Tactic
 * File Name:     OfflineModeBattleScene.cpp
 * File Function: OfflineModeBattleScene类的实现
 * Author:        杨宇琨、刘淑仪、林继申
 * Update Date:   2023/12/31
 * License:       MIT License
 ****************************************************************/
```

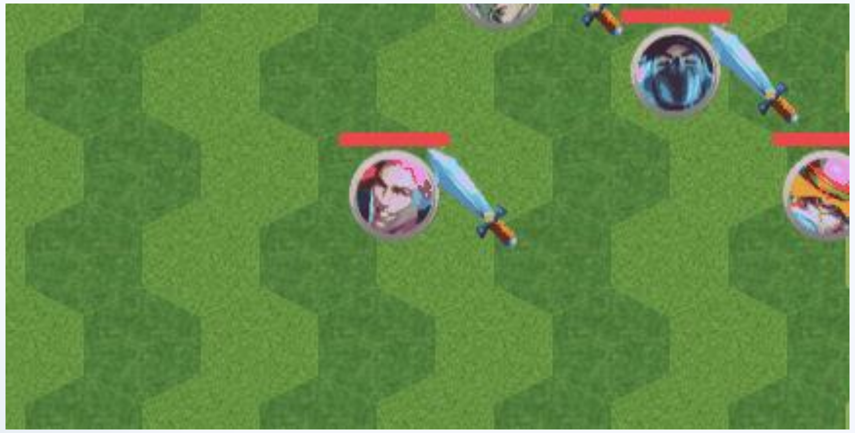
Implementation of Project Features

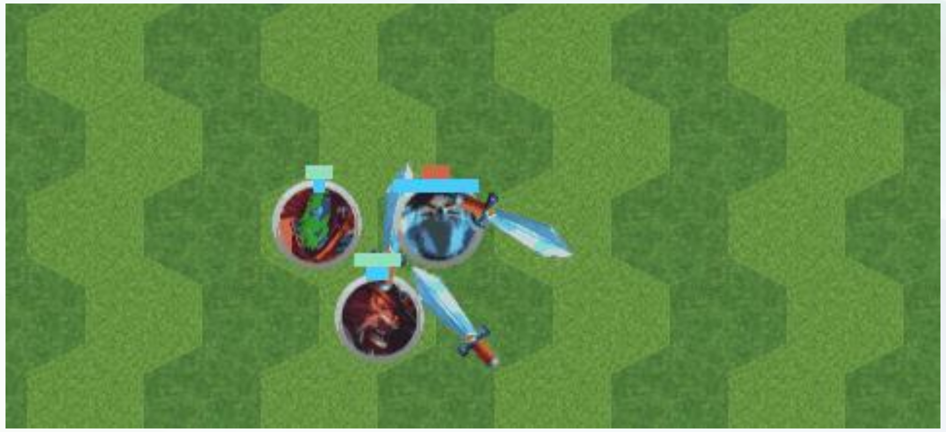
Detailed introduction of features designed in the projects

TONGJI UNIVERSITY

Automatic Battle Logic for Champions

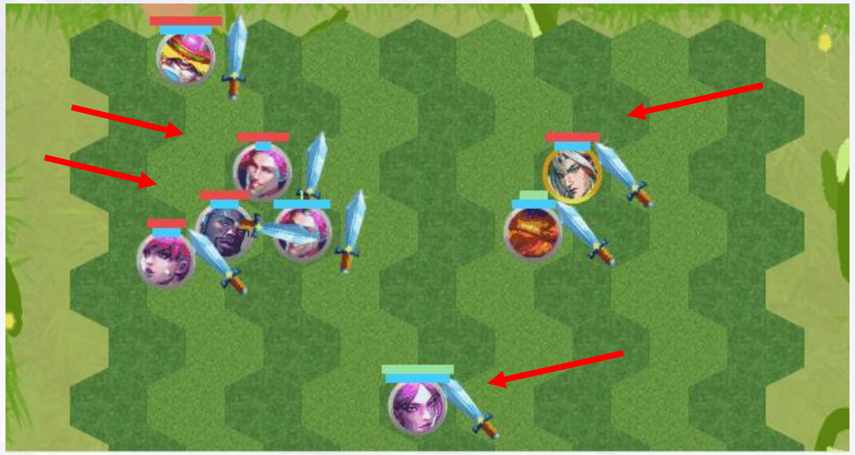
1 Hero's Automatic Enemy Targeting, Walking, and Attacking





2 The Logic for Clearing Actions After Champion's Death

3 Champion skills

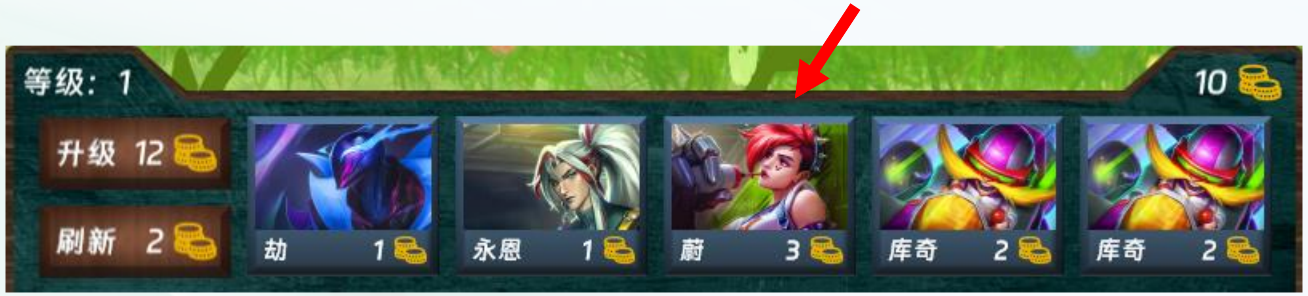


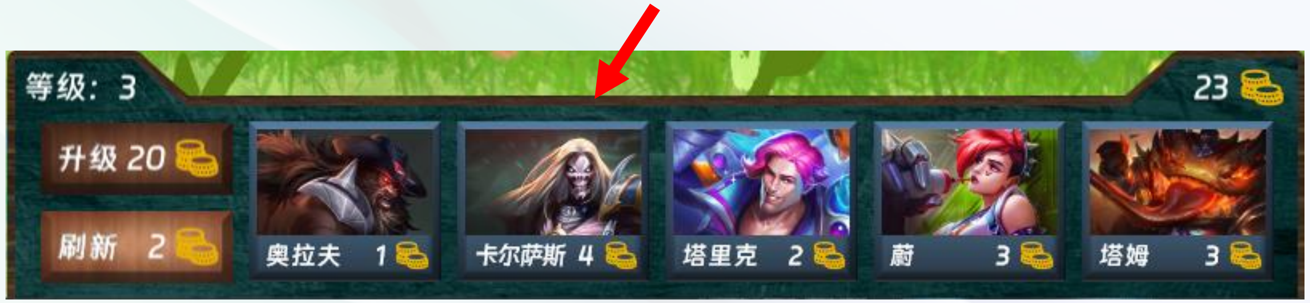
Automatic Battle Logic for Champions

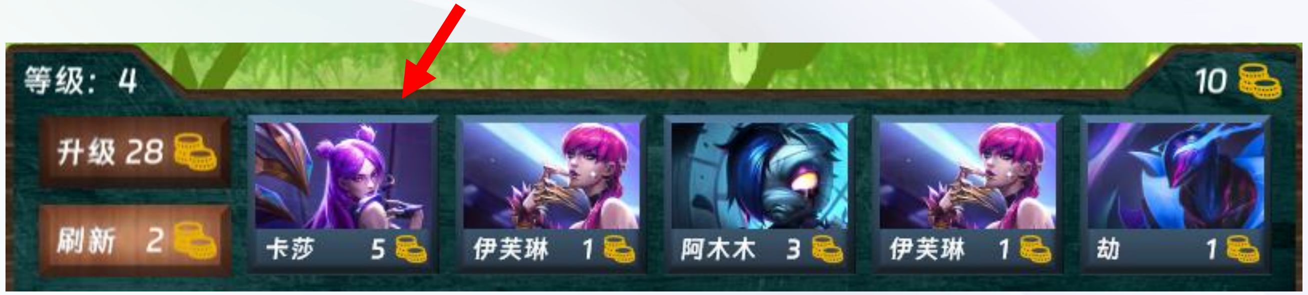
```
// 更新战斗英雄
for (int i = 0; i < battleChampionCount; i++) {
    if (battleChampion[i] != nullptr) { // 存在战斗英雄
        // 设置生命条和经验条
        // 检查角色存活
        if (battleChampion[i]->getAttributes().healthPoints > 0) { // 角色存活
            if (battleChampion[i]->getIsMoving()) { // 角色正在移动
                // 增加移动时间间隔
                if (battleChampion[i]->getMoveIntervalTimer() >= (1.0f /
battleChampion[i]->getAttributes().movementSpeed)) { // 达到停止移动条件
                    // 停止战斗英雄移动
                    // 重置移动时间间隔
                    // 重置战斗英雄移动状态
                    // 重置当前移动
                }
            }
            else { // 角色停止移动
                // 检查是否达到技能触发条件
                if (battleChampion[i]->getAttributes().magicPoints >=
battleChampion[i]->getAttributes().skillTriggerThreshold) {
                    // 触发技能
                }
                // 获取最近敌方战斗英雄
                if (battleChampion[i]->getCurrentEnemy()) { // 获取当前锁定敌人战
斗英雄指针
                    // 攻击范围内存在敌人战斗英雄
                    if (battleChampion[i]->isInAttackRange()) {
                        // 增加攻击时间间隔
                        if (battleChampion[i]->getAttackIntervalTimer() >= (1.0f /
battleChampion[i]->getAttributes().attackSpeed)) { // 达到攻击时间间隔
                            // 攻击
                            // 重置攻击时间间隔
                        }
                    }
                    else { // 攻击范围内不存在敌人战斗英雄
                        // 设置战斗英雄移动状态
                        // 设置当前目标位置
                        // 设置当前移动
                        // 移动武器
                        // 移动生命条
                        // 移动经验条
                        // 移动战斗英雄
                        // 放置战斗英雄
                        // 移除战斗英雄
                        // 设置当前战斗英雄位置
                    }
                }
            }
        }
        else { // 角色死亡
            // 隐藏武器
            // 角色死亡
            // 置空战斗英雄指针
        }
    }
}
```

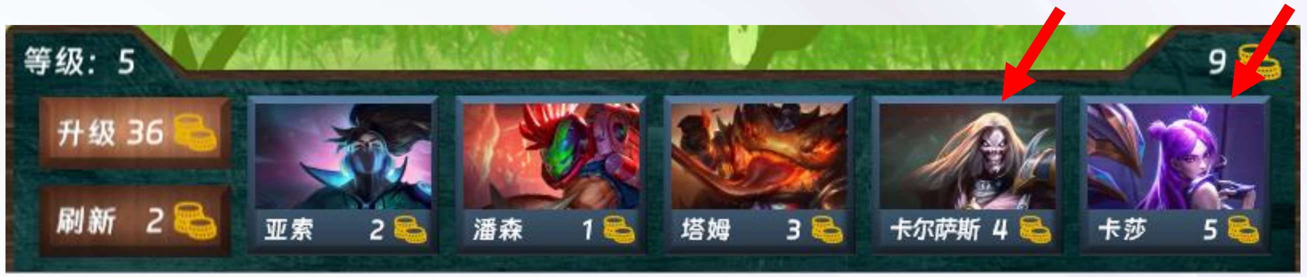
This Loop is Called once per frame

The Shop Recommendation Algorithm

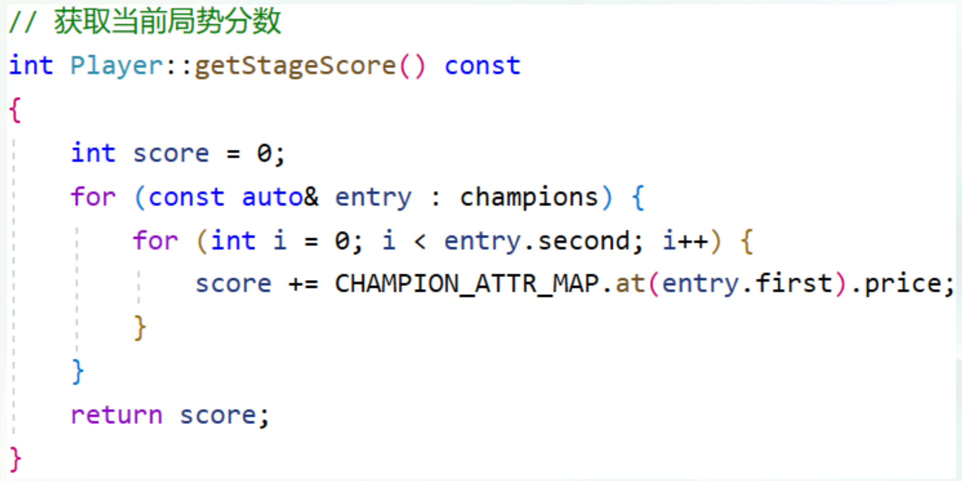






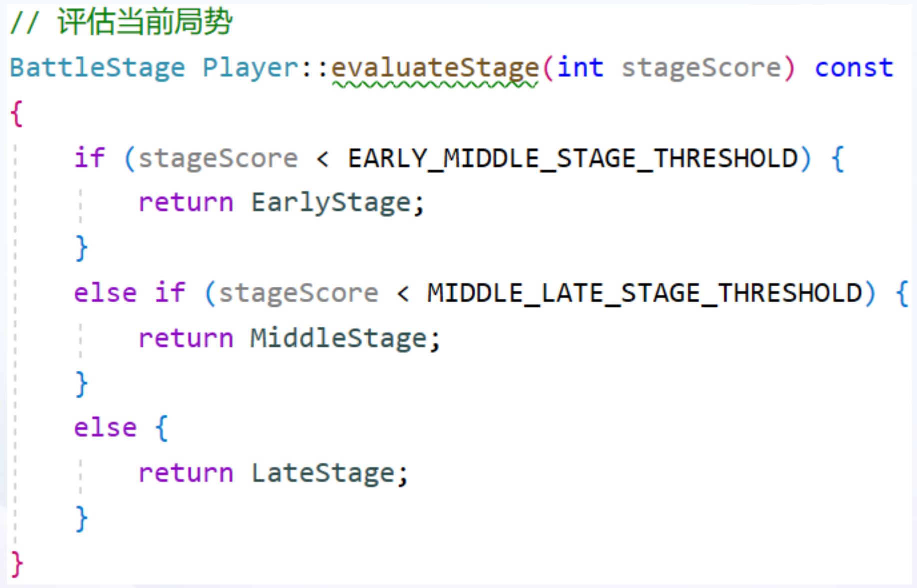


The Shop Recommendation Algorithm



```
// 获取当前局势分数
int Player::getStageScore() const
{
    int score = 0;
    for (const auto& entry : champions) {
        for (int i = 0; i < entry.second; i++) {
            score += CHAMPION_ATTR_MAP.at(entry.first).price;
        }
    }
    return score;
}
```

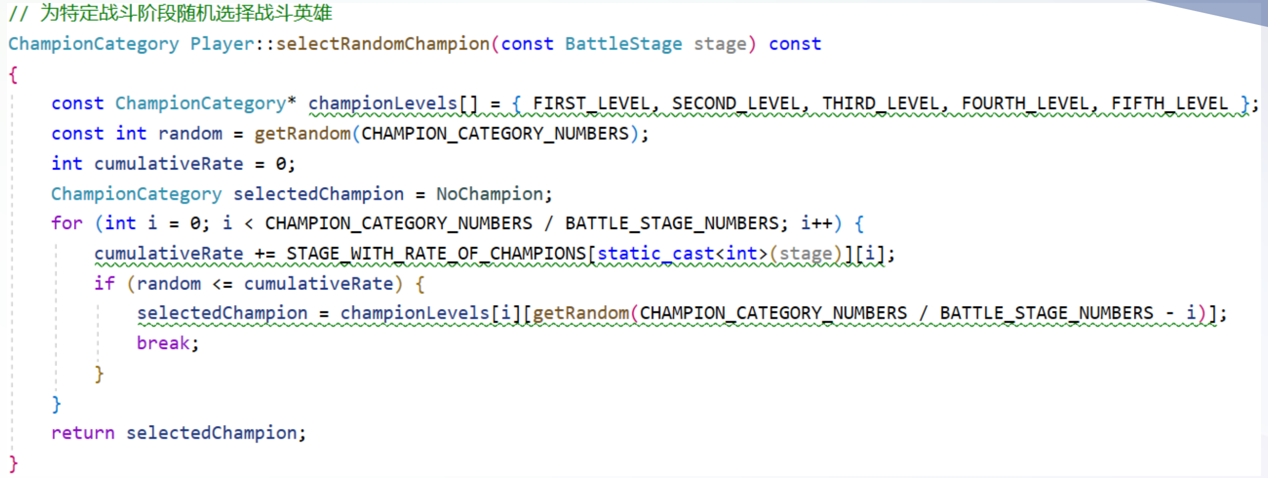
1. Evaluate based on the total value of all heroes



```
// 评估当前局势
BattleStage Player::evaluateStage(int stageScore) const
{
    if (stageScore < EARLY_MIDDLE_STAGE_THRESHOLD) {
        return EarlyStage;
    }
    else if (stageScore < MIDDLE_LATE_STAGE_THRESHOLD) {
        return MiddleStage;
    }
    else {
        return LateStage;
    }
}
```

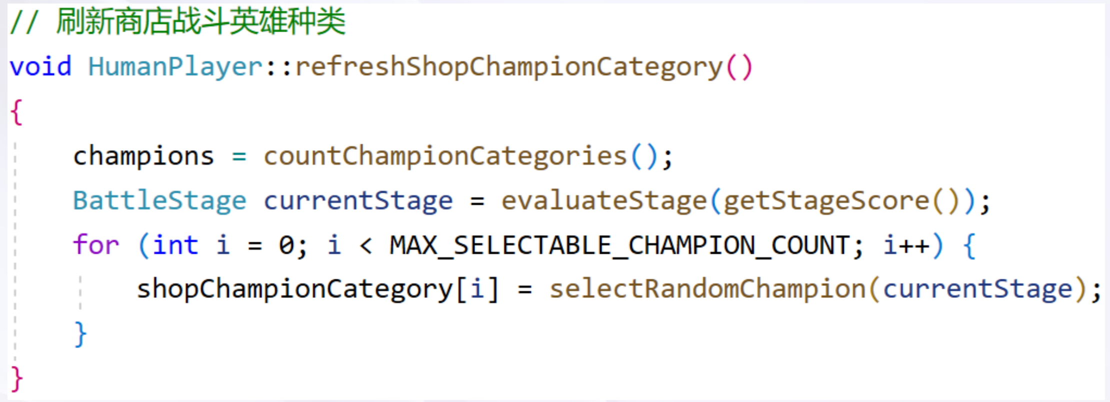
2. Return a Stage

3. Set random number range based on the stage



```
// 为特定战斗阶段随机选择战斗英雄
ChampionCategory Player::selectRandomChampion(const BattleStage stage) const
{
    const ChampionCategory* championLevels[] = { FIRST_LEVEL, SECOND_LEVEL, THIRD_LEVEL, FOURTH_LEVEL, FIFTH_LEVEL };
    const int random = getRandom(CHAMPION_CATEGORY_NUMBERS);
    int cumulativeRate = 0;
    ChampionCategory selectedChampion = NoChampion;
    for (int i = 0; i < CHAMPION_CATEGORY_NUMBERS / BATTLE_STAGE_NUMBERS; i++) {
        cumulativeRate += STAGE_WITH_RATE_OF_CHAMPIONS[static_cast<int>(stage)][i];
        if (random <= cumulativeRate) {
            selectedChampion = championLevels[i][getRandom(CHAMPION_CATEGORY_NUMBERS / BATTLE_STAGE_NUMBERS - i)];
            break;
        }
    }
    return selectedChampion;
}
```

4. Randomly generate heroes for the five slots



```
// 刷新商店战斗英雄种类
void HumanPlayer::refreshShopChampionCategory()
{
    champions = countChampionCategories();
    BattleStage currentStage = evaluateStage(getStageScore());
    for (int i = 0; i < MAX_SELECTABLE_CHAMPION_COUNT; i++) {
        shopChampionCategory[i] = selectRandomChampion(currentStage);
    }
}
```

AI Move-Making Algorithms

EarlyStage

MiddleStage

LateStage

AI Move-Making Algorithms

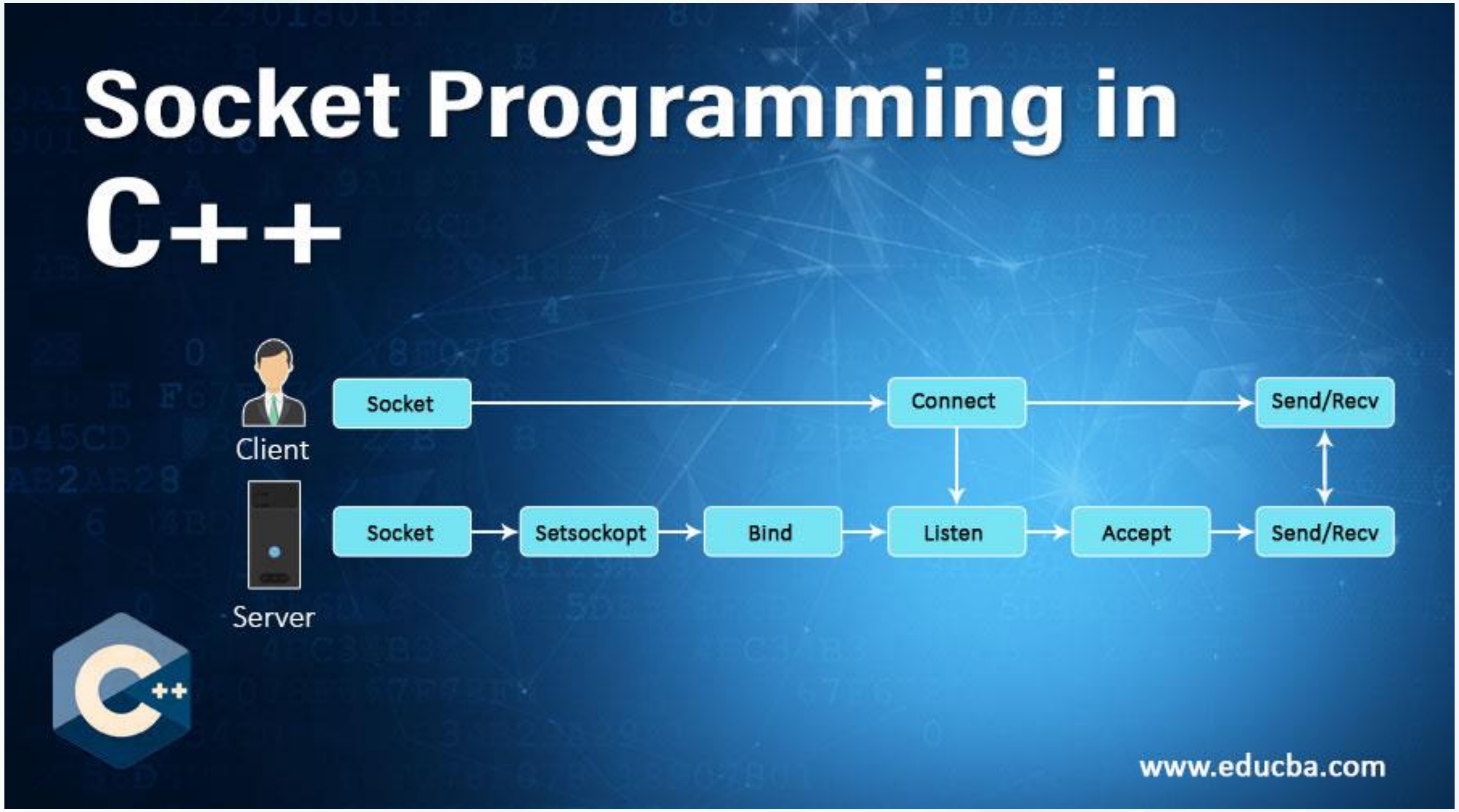
Firstly assess the current owned heroes to determine if purchasing additional heroes would enable an upgrade; if possible, upgrade, otherwise select corresponding heroes based on the current game stage. **(Purchase)**

Classify all heroes into five categories based on their specific strengths, then assign suitable proportions to each category. Determine the hero's strength range of the desired purchase through random number generation and select an appropriate hero from this range using another random number. **(Purchase)**

Establish an overall rating system for each hero to determine their strength based on an overall assessment. Then, considering the current AI battle difficulty, implement varying hero selection strategies. For hard mode, select the top five heroes with the highest ratings; for easy mode, choose the bottom five heroes with the lowest ratings. **(Selection)**

For the selected battle heroes, position arrangement strategies vary based on the current difficulty. In general, specific positions are chosen for rows, while column positions are randomly generated. **(Placement)**

Socket Programming



C++11 New Features

"We are using state-of-the-art tools."

Type Inference

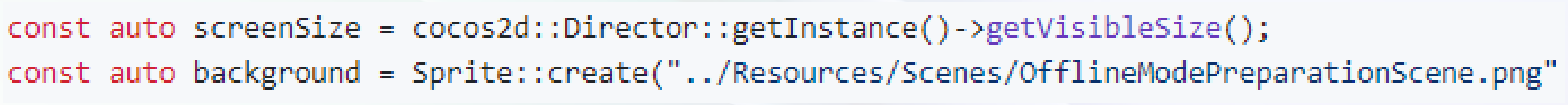
C++11 introduced the auto and decltype keywords, allowing the compiler to automatically deduce the type of variables or expressions, making coding more convenient and code more concise.

auto: Allows the compiler to deduce the type of a variable at compile time, based on the type on the right-hand side of the assignment.

decltype: Unlike auto, which is used for deducing variable types, decltype is used for deducing expression types. It instructs the compiler to analyze the type of an expression without actually evaluating the expression itself.

Type Inference

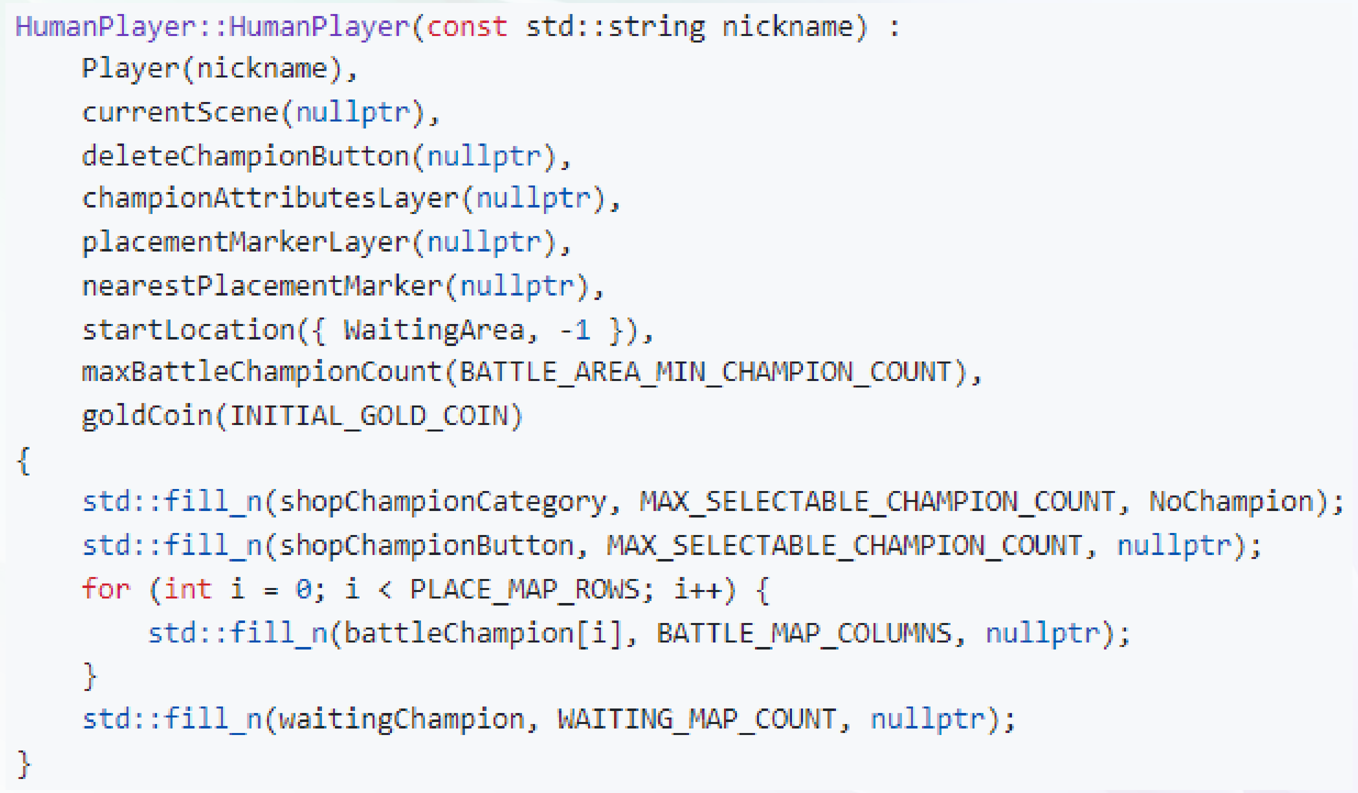
In this project, auto and decltype were used for creating variables when dealing with complex data types. This approach facilitated handling complex type expressions, reduced bugs caused by type errors, and improved code maintainability. Examples of their specific usage are provided below.



```
const auto screenSize = cocos2d::Director::getInstance()->getVisibleSize();
const auto background = Sprite::create("../Resources/Scenes/OfflineModePreparationScene.png"
```

List Initialization

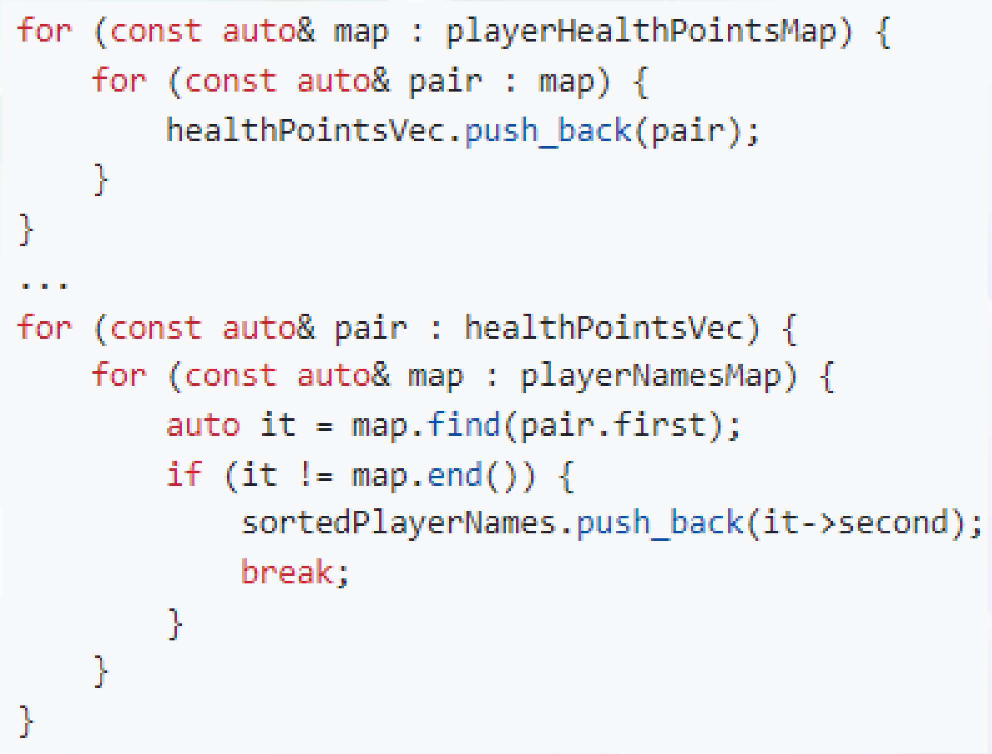
C++11 introduced list initialization, also known as uniform initialization, which is a new syntax using curly braces { } to initialize objects. It offers a more consistent, safer way of initialization and can be used for almost all types of initialization, including primitive data types, objects, arrays, containers, and more.



```cpp
HumanPlayer::HumanPlayer(const std::string nickname) :
    Player(nickname),
    currentScene(nullptr),
    deleteChampionButton(nullptr),
    championAttributesLayer(nullptr),
    placementMarkerLayer(nullptr),
    nearestPlacementMarker(nullptr),
    startLocation({ WaitingArea, -1 }),
    maxBattleChampionCount(BATTLE_AREA_MIN_CHAMPION_COUNT),
    goldCoin(INITIAL_GOLD_COIN)
{
    std::fill_n(shopChampionCategory, MAX_SELECTABLE_CHAMPION_COUNT, NoChampion);
    std::fill_n(shopChampionButton, MAX_SELECTABLE_CHAMPION_COUNT, nullptr);
    for (int i = 0; i < PLACE_MAP_ROWS; i++) {
        std::fill_n(battleChampion[i], BATTLE_MAP_COLUMNS, nullptr);
    }
    std::fill_n(waitingChampion, WAITING_MAP_COUNT, nullptr);
}
```

Range-based for loop

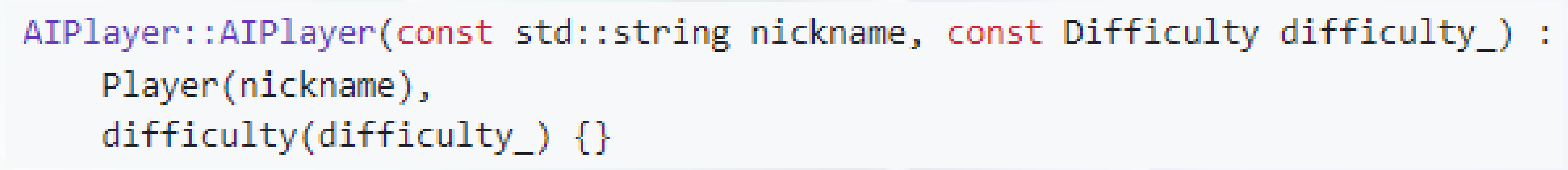
C++11 introduced the range-based for loop, a new looping syntax that simplifies and makes iterating over containers (such as arrays, vectors, lists, etc.) and ranges more straightforward. This loop automatically iterates through each element in a container or range without the need for manually managing iterators or indices.



```
for (const auto& map : playerHealthPointsMap) {
    for (const auto& pair : map) {
        healthPointsVec.push_back(pair);
    }
}
...
for (const auto& pair : healthPointsVec) {
    for (const auto& map : playerNamesMap) {
        auto it = map.find(pair.first);
        if (it != map.end()) {
            sortedPlayerNames.push_back(it->second);
            break;
        }
    }
}
```

Inherited Constructors

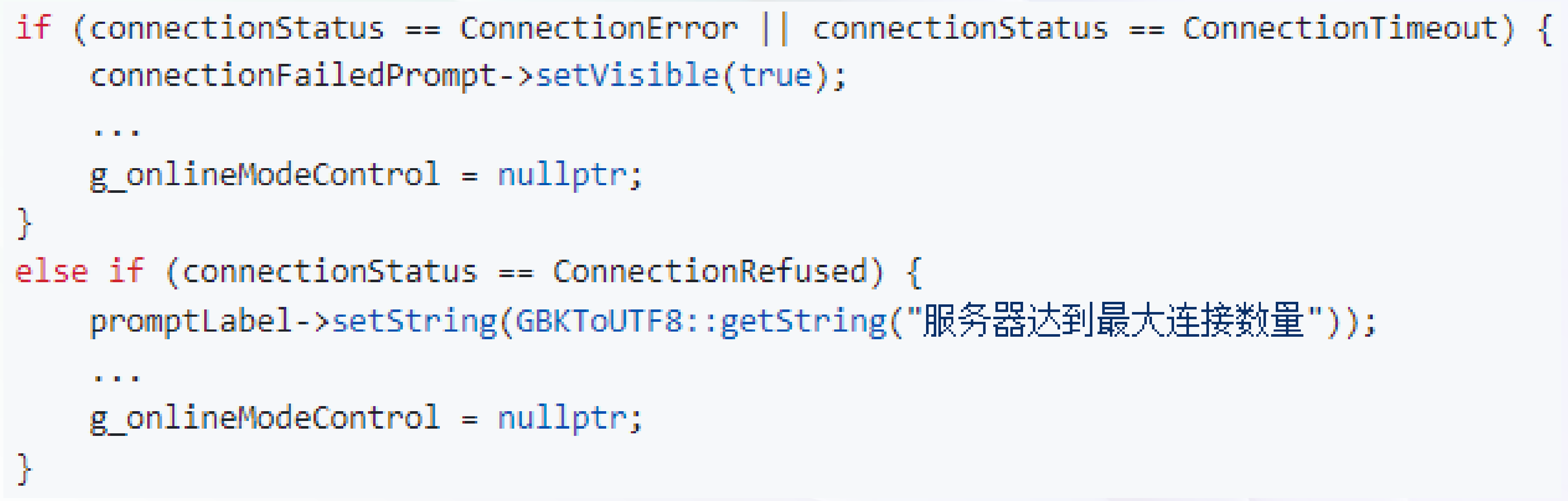
In C++, Inheriting Constructors is a feature introduced in the C++11 standard, allowing derived classes to inherit the constructors of their base class. This feature primarily addresses the issue of redundant code in derived classes by eliminating the need to rewrite code identical to the base class constructors, thereby enhancing code maintainability and conciseness.



```cpp
AIPlayer::AIPlayer(const std::string nickname, const Difficulty difficulty_) :
    Player(nickname),
    difficulty(difficulty_) {}
```

nullptr Keyword

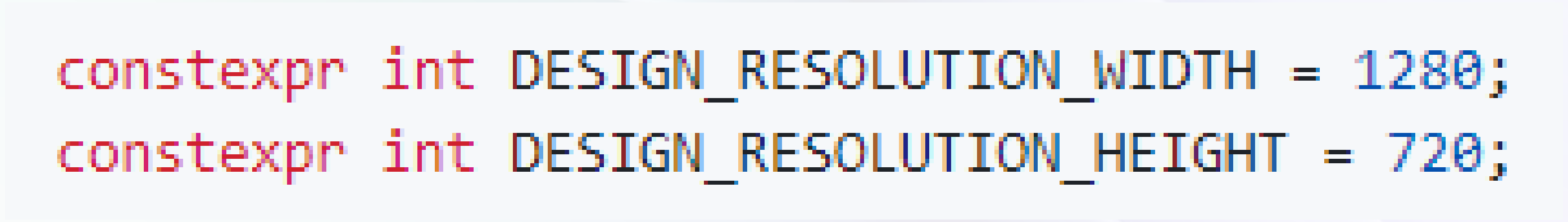
C++11 introduced the nullptr keyword, which is a literal representing a null pointer. It serves as a replacement for the traditional NULL used in previous C++ standards. In earlier standards, NULL was often defined as 0 or ((void*)0), which could lead to type ambiguities and some hard-to-detect errors. The introduction of nullptr addresses these issues and provides a clearer and safer way to represent a null pointer.



```
if (connectionStatus == ConnectionError || connectionStatus == ConnectionTimeout) {
    connectionFailedPrompt->setVisible(true);
    ...
    g_onlineModeControl = nullptr;
}
else if (connectionStatus == ConnectionRefused) {
    promptLabel->setString(GBKToUTF8::getString("服务器达到最大连接数量"));
    ...
    g_onlineModeControl = nullptr;
}
```

constexpr Keyword

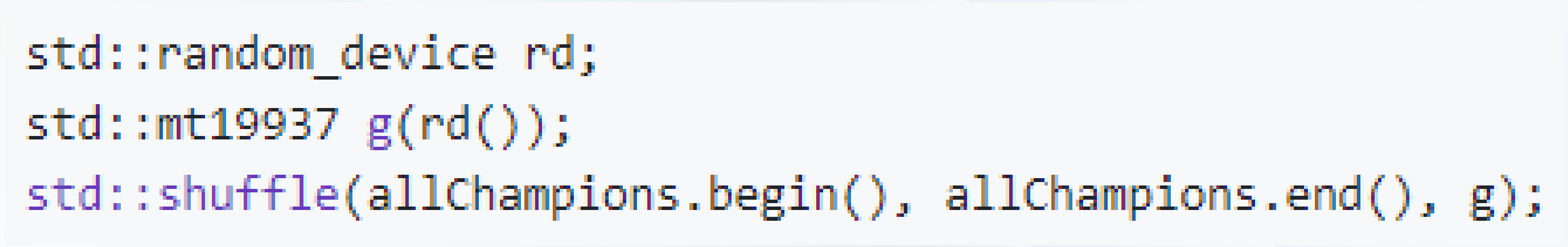
constexpr is a keyword introduced in C++11 for compile-time constants and constant functions. Variables or functions marked with constexpr are true constants that are computed at compile time and cannot be altered during runtime. constexpr can be applied to functions, and the return value of such functions is computed as a constant at compile time whenever possible. However, if the function cannot be evaluated at compile time, it is treated as a regular function.



```
constexpr int DESIGN_RESOLUTION_WIDTH = 1280;
constexpr int DESIGN_RESOLUTION_HEIGHT = 720;
```

New Random Number Generation Library

C++11 introduced a comprehensive random number generation library that offers various random number generators (RNGs) and probability distributions. This library's design aims to address the limitations of the random number functionalities in older C++ standards, providing a more flexible, efficient, and type-safe approach to generating random numbers.



```
std::random_device rd;
std::mt19937 g(rd());
std::shuffle(allChampions.begin(), allChampions.end(), g);
```

The chrono Library and thread Library's this_thread Namespace

C++11 introduced the chrono library for time handling, providing representations for time points, durations, and clocks.

Additionally, the C++11 standard introduced the <thread> library, which allows control and management of threads. Within this library, std::this_thread is a namespace that offers functions related to the current thread. Particularly, std::this_thread::sleep_for is a function used to make the current thread sleep for a specified duration of time.

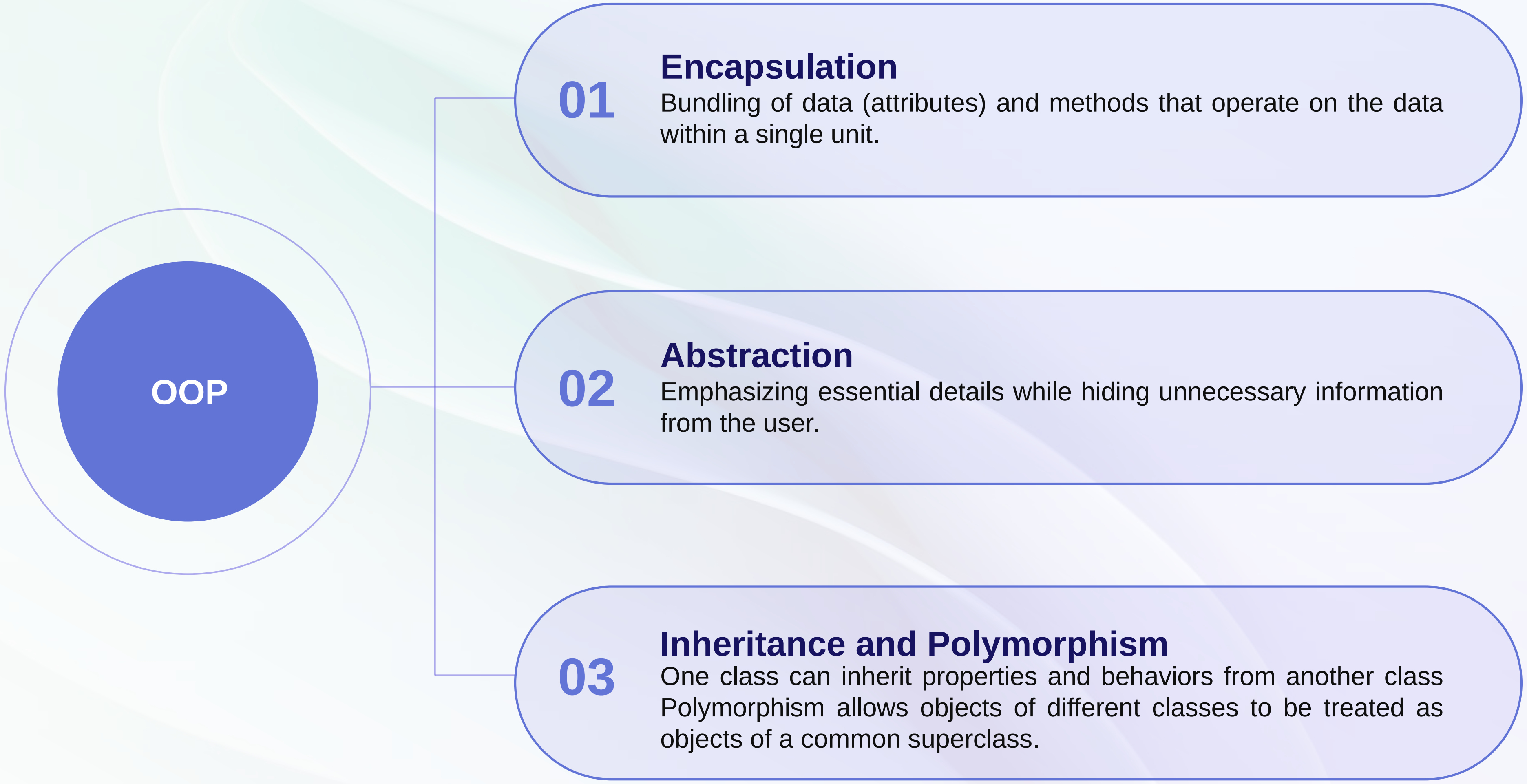
Unicode

In C++11, support for Unicode was introduced along with several new string literals. In this project, the u8 prefix is utilized to denote a string literal in UTF-8 encoding. UTF-8 is a variable-width character encoding used for encoding Unicode characters. This facilitates C++ programs to handle multiple languages and character sets more seamlessly.

Conclusion and Review

Understanding of OOP principles
and future feature expansion

Fundamental Principles



01 **Encapsulation**
Bundling of data (attributes) and methods that operate on the data within a single unit.

02 **Abstraction**
Emphasizing essential details while hiding unnecessary information from the user.

03 **Inheritance and Polymorphism**
One class can inherit properties and behaviors from another class Polymorphism allows objects of different classes to be treated as objects of a common superclass.

Usage of Pointer

1. **Dynamic Object Allocation and Management**: In OOP languages, objects are often dynamically allocated using new (in C++) or similar methods. Pointers are used to store the addresses of these objects, allowing for dynamic creation, access, and management of objects.
2. **Implementation of Polymorphism**: Polymorphism is a key concept in OOP that allows objects of different classes to be treated as instances of a common superclass. Pointers or references are commonly used to point to different objects of various classes, facilitating polymorphic behavior through virtual functions or interfaces.
3. **Object Association and Reference**: OOP allows objects to reference or associate with each other. Pointers facilitate the creation of complex relationships and data structures, such as linked lists, trees, etc., by enabling objects to reference one another.
4. **Enhancing Program Performance**: In certain scenarios, using pointers can enhance program performance as they enable direct memory access without the overhead of copying data. This can be advantageous for large data structures or situations requiring frequent memory access.
5. **Memory Management and Resource Deallocation**: Objects allocated on the heap need to be manually deallocated to prevent memory leaks. Pointers are used to track dynamically allocated memory and enable the release of memory using delete operations when the objects are no longer needed.

Future Plan

01 Online Mode

02 UI Design

03 Champion Diversity

同濟大學
TONGJI UNIVERSITY

2024 - 01 - 02

THANK YOU

2250758 林继申 (组长)
2251730 刘淑仪
2252712 杨兆镇
2252843 杨宇琨

GitHub: https://github.com/MinmusLin/Teamfight_Tactics
Speaker: 杨宇琨 林继申