

学号：22200107115

成绩：

苏州科技大学

毕业设计（论文）

题 目 教育游戏软件的三维地图交互引擎设计与实现

性 质：毕业设计☒

毕业论文☐

院 系：电子与信息工程学院

专 业：计算机科学与技术

年 级：2022 级

姓 名：赵乐

指导教师：奚雪峰

二〇二六年

教育游戏软件的三维地图交互引擎设计与实现

摘 要

传统在线学习和题库式练习，一般还是把分数、正确率作为主要反馈，学生如果长时间重复刷题，很容易出现疲劳感，学习动力也会慢慢下降。相比这种方式，游戏在目标推进、即时反馈和阶段奖励方面有比较明显的优势，能让学习过程有更强的参与感。

围绕教育游戏中地图结构比较固定、难度调节不够灵活、学习行为和游戏反馈联系不够紧密等问题，本文设计并实现了一个面向三维地图交互的教育游戏引擎系统。该系统以节点式拓扑地图组织关卡路径，在三维场景中呈现路线、节点和关卡入口。系统会把答题正确率、通关率、学习投入、习惯得分，以及错误和超时惩罚等数据纳入学力值计算，再依据学力值生成复杂程度不同的关卡地图。学生在关卡中通过答题来控制小车战斗，答题结果会影响攻击、扣血、奖励和结算，相关学习行为也会回传到后端，用于后续学力值更新。

从实现结果看，本课题完成了自适应地图生成、题目战斗、奖励成就和学习数据回传等核心流程。在不同学力值条件下，地图路线数量、Boss 节点数量和奖励节点数量会出现比较明显的差异，验证了学习行为驱动地图交互机制的可行性。

关键词 教育游戏；动态难度调整；学习行为数据；Unity

Design and Implementation of a Three-Dimensional Map Interaction Engine for Educational Game Software

Abstract

Traditional online learning platforms and question-bank practice systems usually take scores and answer accuracy as the main form of feedback. This kind of feedback is clear, but when students repeat similar exercises for a long time, the learning process can become monotonous, and their motivation may also decline. Compared with simple question practice, games can provide goal progression, timely feedback, staged rewards, and a more continuous sense of participation. These features make educational games useful for improving the learning experience.

Focusing on the problems of fixed map structures, limited difficulty adjustment, and weak connections between learning behaviors and game feedback in educational games, this thesis designs and implements a three-dimensional map interaction engine for educational game software. The system uses a node-based topological map to organize level paths, and presents routes, nodes, and level entrances in a three-dimensional scene. It calculates the learning ability score according to answer accuracy, level completion rate, learning engagement, habit score, and penalties caused by wrong answers or timeout. Based on this score, the system generates level maps with different degrees of complexity. In each level, students answer questions to control the vehicle battle. The answer result will affect attacks, health deduction, rewards, and settlement, while related learning behavior data is uploaded to the back end for later updating of the learning ability score.

The implemented prototype completes the main processes of adaptive map

generation, question-based battle, reward and achievement feedback, and learning data uploading. Under different learning ability scores, the number of map routes, Boss nodes, and reward nodes shows clear changes. The test results indicate that the proposed method can connect learning behavior data with three-dimensional map interaction, and provides a feasible implementation scheme for personalized learning paths and game-based feedback in educational games.

Keywords Educational Game; Dynamic Difficulty Adjustment(DDA); Learning Behavior Data; Unity

目 录

第 1 章 绪论	1
1.1 引言	1
1.2 研究背景与意义	1
1.3 国内外研究现状	3
1.3.1 游戏化学习与教育游戏研究现状	3
1.3.2 游戏化学习对动机、投入和反馈的影响	3
1.3.3 动态难度调整与心流体验研究现状	4
1.3.4 程序化内容生成与地图关卡生成研究现状	4
1.4 本文研究内容	5
1.5 论文组织结构	6
第 2 章 相关技术与理论	8
2.1 系统技术路线概述	8
2.2 游戏化学习	8
2.3 动态难度调整	9
2.4 程序化地图生成	10
2.5 Unity 客户端开发技术	10
2.6 Spring Boot 后端与数据服务技术	11
2.7 MySQL 数据库与数据交互技术	11
2.8 本章小结	12
第 3 章 需求分析	13
3.1 可行性分析	13
3.1.1 技术可行性分析	13
3.1.2 经济可行性分析	13
3.1.3 操作可行性分析	14
3.2 需求分析	14
3.2.1 系统需求概述	14
3.2.2 用户角色分析	15
3.2.3 功能需求分析	15
3.2.4 非功能需求分析	16
3.2.5 核心用例分析	18
3.3 本章小结	23
第 4 章 系统总体设计	24
4.1 系统总体设计	24
4.2 模块交互流程设计	24
4.3 学力值更新模型设计	25
4.4 难度映射与地图交互设计	27

4.5 节点式地图生成算法设计	27
4.6 本章小结	29
第 5 章 系统实现	30
5.1 系统实现环境与职责划分	30
5.2 Unity 玩家端实现	30
5.2.1 地图配置数据实现	32
5.2.2 节点地图生成实现	32
5.2.3 题目适配与降级处理实现	33
5.2.4 节点访问与进度保存实现	33
5.3 Spring Boot 后端实现	34
5.4 MySQL 数据库实现	34
5.5 学力值更新算法实现	36
5.6 题目战斗与统计实现	38
5.7 成就与游戏结果回传实现	39
5.8 Vue 管理后台实现	40
5.9 本章小结	41
第 6 章 系统测试	42
6.1 测试环境	42
6.2 测试用例设计	42
6.3 系统运行结果展示	44
6.3.1 学生端运行结果展示	44
6.3.2 管理端运行结果展示	48
6.4 测试结果分析	49
6.5 本章小结	50
第 7 章 结论	51
致 谢	53
参 考 文 献	54
附录 A 译文	56
附录 B 外文原文	77

第1章 绪论

1.1 引言

随着教育信息化建设不断推进，网络题库、在线测试和学习平台已经成为学生日常学习中比较常见的工具。这类系统能够记录学习时长、答题正确率、知识点完成情况、错误分布和任务进度等数据，也为个性化学习提供了基础。不过，在不少实际应用里，这些数据更多还是用于成绩统计或学习报告展示，学习反馈也往往停留在分数、排名和正确率等结果性信息上。对于需要长期坚持的知识巩固任务来说，如果只是反复刷题，学习体验容易变得单调，反馈强度不够，持续学习的动力也会受到影响。

游戏本身有明确目标、即时反馈、阶段奖励、难度递进和持续成长等机制，玩家在完成某个行为之后，通常能比较快地感知到结果。已有游戏化学习研究也表明，把游戏元素引入学习活动时，学习目标、反馈机制和学习者动机之间的关系，会直接影响学习体验^{[1][2]}。因此，把这些机制合理放进学习场景中，可以让学生在完成学习任务的同时，得到更连续的过程反馈。教育游戏的价值不只是把题目包装成游戏界面，更重要的是让学习行为真正影响游戏进程，使答题、通关、奖励和成长之间有实际联系。

在游戏系统中，地图是组织任务和引导玩家行为的重要载体。程序化内容生成研究指出，地图、关卡、场景和任务等游戏内容，可以在一定规则约束下通过算法生成，从而提升内容多样性和可重复游玩性^{[10][14]}。节点式地图可以通过路线分支、节点类型、Boss 挑战和奖励节点来表达不同风险与收益，也比较适合承载阶段化学习任务。

1.2 研究背景与意义

在传统教育软件和学校日常训练中，学习任务通常会以章节、题目或测试的形式呈现，学习者按照固定顺序完成任务。这种方式结构清楚，教师也比较容易检查学习结果，用于知识点巩固和阶段性测验时比较合适。从实际学习过程看，如果任务顺序一直固定，反馈方式也只是给出一个结果，系统就不容易表现出不同学生之间的能力差别，也不方便根据学生当时的表现继续调整后面的任务。

教育游戏给学习反馈提供了另一种处理方式。相关综述中提到，游戏化课堂里常

见的元素有分数、徽章、排行榜和关卡等，这些内容会在一定程度上影响学习投入、满意度和内在动机^[2]。在教育游戏中，关卡挑战、地图探索、角色成长、奖励和成就等内容，可以把比较抽象的学习结果变成学生能看到的变化。比如答题正确以后，系统可以给出攻击或通关反馈，连续学习以后可以解锁成就，正确率提高以后也可以进入难一点的路线。这些反馈可以让学生比较直观地看到自己的学习进度，也能对持续学习起到一定帮助。

不过，现有一些教育游戏虽然使用了积分、徽章、排行榜和闯关方式，但地图结构和难度变化还不够灵活。一方面，固定地图开发起来比较方便，但不太适合能力差别较大的学生；另一方面，学习行为数据很多时候只用于结果展示，没有真正参与关卡结构和难度调整。动态难度调整研究认为，系统可以根据玩家表现和游戏过程状态来改变后续挑战强度，让任务难度和用户能力保持相对匹配^{[6][7][8]}。因此，在教育游戏中，需要设计一种能根据学习行为调整地图结构和关卡难度的机制。

本文的研究意义主要体现在三个方面。首先，在教育游戏设计层面，本文将学习行为数据与节点式地图生成相结合，使地图从静态关卡列表转变为能够反映学习状态的动态交互媒介。其次，在学习激励层面，本文将题目练习、战斗反馈、道具奖励和成就反馈整合起来，使学生能够直观看到学习行为带来的游戏变化。最后，在工程实践层面，本文基于 Unity、Spring Boot 和 MySQL 实现原型系统，为教育游戏中自适应地图生成和学习行为反馈提供一种可参考的设计方案。

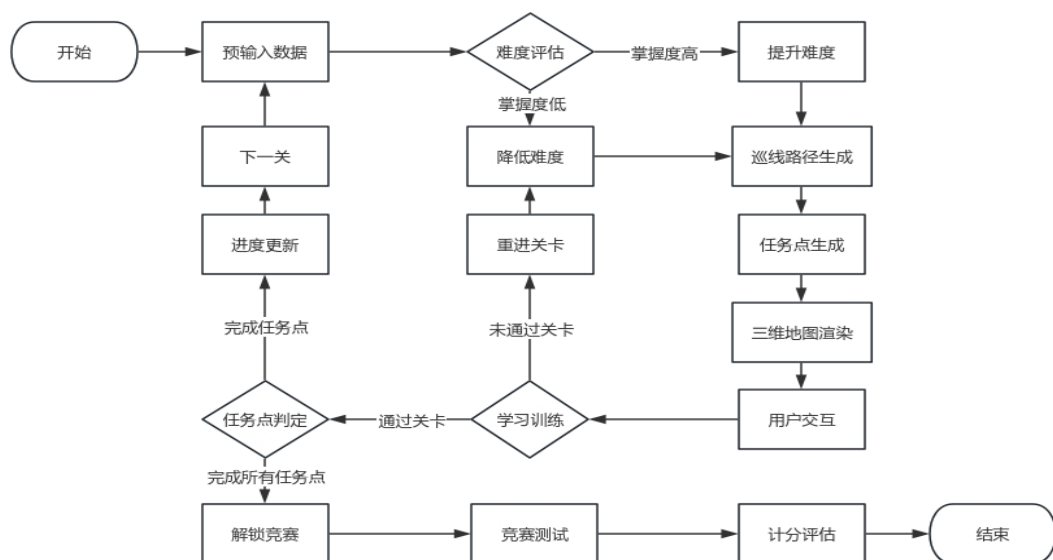


图 1-1 教育游戏学习行为闭环示意图

如图 1-1 所示，系统围绕“学习行为数据采集—学力值计算—地图参数生成—Unity 地图展示—答题战斗—奖励成就反馈—行为数据回传”形成闭环。学习行为不只用于结果统计，而是会继续影响下一轮地图结构和关卡难度，从而体现系统的自适应特征。

1.3 国内外研究现状

1.3.1 游戏化学习与教育游戏研究现状

游戏化学习和教育游戏研究主要关注如何将游戏中的目标、规则、反馈、挑战 and 奖励等机制引入学习活动，使学习者在完成知识训练的同时获得更强的参与感。顾怡雯等对 2015 年至 2019 年国际游戏化学习实证研究进行内容分析后指出，相关研究热点主要集中在学习特征分析、游戏元素设计、教学工具应用效果和数字化游戏学习工具评估等方面^[1]。这也说明，游戏化学习研究已经不再只是给学习任务加一层娱乐化外观，而是开始进一步关注游戏机制、学习者特征和教学目标之间的关系。

从具体学科应用来看，基于游戏的学习已经在数学、语言等学科中积累了一定数量的实证研究。李佳平等对 2016 年至 2021 年基于游戏的数学学习实证研究进行了系统综述，认为相关研究既关注知识点掌握，也关注学习者情感态度和学习动机^[3]。郑春萍等梳理了近十年数字游戏应用于语言教学的研究，指出数字游戏能为学习者提供情境、互动和反馈，但其教育应用仍需要围绕教学目标来设计^[4]。朱玉林等面向初中化学概念学习开发了“化学飞车”教育游戏，把化学概念判断和赛车操作、加速反馈、错误惩罚、错题整理结合起来，并通过实验验证该游戏对化学概念学习有促进作用^[18]。Rose 等在高等教育场景中评估了教育游戏替代传统作业的效果，发现教育游戏能够在不降低学习成绩的情况下保持较高的学生参与度^[19]。这些研究说明，教育游戏的价值不仅在于提高学习活动的趣味性，还在于通过任务反馈、情境交互和过程记录强化学习行为。这些研究为本文将题目练习、关卡挑战和地图推进结合起来提供了背景依据。

1.3.2 游戏化学习对动机、投入和反馈的影响

游戏化学习能否发挥作用，关键不只在是否使用游戏元素，还在于游戏反馈能否与学习行为形成有效联系。白书瑞、丘琪鸿对游戏化课堂中的学生动机学习结果进行系统文献综述，比较游戏化课堂和传统课堂后指出，游戏化学习有助于提高学生的

课程投入程度、满意度和内在动机，但其对自我效能感的提升效果相对有限；同时，排行榜、分数、徽章和关卡仍是最常见的游戏设计元素^[2]。这说明游戏化设计确实可能提升学习参与，但也存在机制单一、反馈停留在外部奖励层面的风险。

教育游戏中的反馈机制还与心流体验密切相关。孙竹墨等以《我的世界：教育版》中的编程任务为实验场景，研究教育游戏中心流体验的形成机制，结果显示挑战—技能平衡和临场感均会正向影响心流体验，心流体验又会对学习满意度产生正向作用^[5]。因此，在教育游戏系统设计中，应尽量避免任务过难导致挫败，也要避免任务过易导致无聊。本文将答题结果与攻击、扣血、结算和奖励反馈绑定，目的正是使学习行为能够即时转化为游戏结果，并通过关卡推进和成就反馈维持学习过程中的目标感。

1.3.3 动态难度调整与心流体验研究现状

动态难度调整是游戏根据玩家状态自动调整挑战强度的一类方法，其目标是使玩家保持在较合适的挑战区间。李通围绕数字化游戏动态难度调整模型进行研究，强调应根据玩家表现和游戏过程状态对难度进行调节^[6]。王广新、刘聪关于学生对动态难度游戏体验的研究也说明，动态难度条件会影响学习者的享乐体验和成就目标取向^[7]。在教育游戏中，动态难度调整有其特殊意义。难度变化不能只追求游戏刺激，还需要服务于知识巩固、学习保持和能力提升。

已有教育游戏研究已经开始把学习表现和难度自适应结合起来。新田耕豊基于《成语小秀才》的通关数据设计了难度自适应机制，通过分析玩家通关表现，为玩家提供个性化难度关卡，从而改善玩家学习兴趣和参与表现^[8]。Lora-Ariza 等则把动态难度调整和心流理论结合起来，在俄罗斯方块场景中根据玩家技能水平和游戏状态调整难度，并通过实验检验其对心流感知的影响^[9]。这些研究说明，动态难度调整可以作为连接学习表现和游戏挑战的一种重要方法。本文在此基础上使用学力值描述学生近期学习状态，并将学力值映射到地图复杂度和题目难度中。

1.3.4 程序化内容生成与地图关卡生成研究现状

程序化内容生成是游戏研究中的重要方向，常用于自动生成地图、关卡、场景、任务和道具等内容。王鹏云围绕基于 PCG 的关卡生成游戏系统进行设计与实现，说明 PCG 方法能够在规则约束下生成具有差异性的关卡内容，从而降低人工配置成本并提高游戏内容变化性^[17]。Wattanapornprom 等将进化算法用于 2.5D Rogue-Lite 游戏

的程序化内容生成，体现了 PCG 方法在关卡结构变化和重复游玩体验构建中的应用价值^[20]。Hendrikx 等对游戏程序化内容生成进行综述，将游戏内容划分为多个层次，并指出程序化生成可以缓解人工内容生产成本高和内容规模扩展困难等问题^[10]。Summerville 等进一步总结了基于机器学习的程序化内容生成方法，认为 PCG 已经可以面向平台关卡、地图、交互叙事和卡牌等功能性游戏内容展开生成^[11]。Smith 则从未来发展角度指出，程序化内容生成的价值不仅在于提高可重复游玩性，也在于支持设计控制和适配性体验^[11]。

在具体地图和场景生成方面，国内研究也关注了规则约束下的游戏内容生成和 Roguelike 类地图设计。靳仕恒等研究面向游戏场景生成的细分插槽 WFC 算法，体现了在规则约束下自动生成游戏场景的思路^[13]。郑爱宇等基于 Unity3D 实现 RougeLike-DBG 风格游戏，并涉及随机地图生成、战斗系统和路径寻路等关键技术^[14]。这些研究为本文采用节点式地图组织关卡提供了参考。与开放世界或复杂三维地形生成相比，节点式拓扑地图更适合教育游戏原型系统，因为它既能表达路线选择、Boss 挑战和奖励节点，又便于将学习行为数据映射为路线数量、节点类型和关卡难度。

综合上述研究可以看出，游戏化学习、动态难度调整和程序化内容生成分别从学习激励、难度适配和内容生成角度为教育游戏设计提供了理论与方法基础。但现有研究较多分别讨论学习动机、游戏反馈、难度调节或地图生成本身，对“学习行为数据—地图结构变化—题目战斗反馈—学习数据回传”的闭环结合仍有进一步探索空间。本文的研究重点正是将学生答题表现、通关结果、学习投入等行为数据转化为学力值，并进一步驱动节点式地图复杂度、Boss 节点和奖励节点变化，使学习行为不只用于结果统计，而是直接参与教育游戏地图交互过程。

1.4 本文研究内容

本文围绕教育游戏地图交互引擎的设计与实现展开研究。结合当前原型系统的实际形态，本文将系统划分为学习数据服务层、地图拓扑生成层和 Unity 交互表现层。学习数据服务层负责记录用户答题行为、关卡耗时、通关状态和奖励结果，并根据这些数据计算学力值；地图拓扑生成层负责根据学力值生成不同复杂度的节点地图；Unity 交互表现层负责展示地图、响应节点点击、加载关卡、执行小车战斗、展示奖励和成就树。

本文主要研究内容如下：

(1) 学习行为数据建模。本文会分析教育游戏场景中可以采集的学习行为，包括答题正确率、答题耗时、超时次数、错误次数、关卡胜负、Boss 击败数量和连续学习情况，并把这些行为整理为学力值计算所需要的指标。

(2) 学力值计算方法设计。系统需要先建立一个学力值计算模型，用来表示学习者当前的大致能力状态。该模型把不同学习行为按一定权重合并成一个数值，再根据数值所在区间，把学习者划分为低、中、高等不同能力等级。

(3) 基于学力值的地图生成算法设计。系统根据学力值改变地图路线数量、分支情况、Boss 节点数量、奖励节点数量和题目难度，使生成出来的地图能够反映学习者之间的能力差别。

(4) 题目驱动的小车战斗机制实现。系统把题目作答过程和小车战斗反馈联系起来，用户答对题目时可以攻击敌人，答错或超时时会扣减小车生命值，这样学习结果就会直接影响游戏流程。

(5) 奖励与成就反馈机制实现。系统设置道具奖励和成就树，把关卡胜利、连续学习、正确率提升、Boss 击败等行为转成可见的成长反馈。

(6) 原型系统集成与测试。本文基于 Unity、Spring Boot 和 MySQL 实现原型系统，并通过功能测试、地图生成测试、接口测试和流程测试验证系统可行性。

1.5 论文组织结构

本文共分为七章。

第 1 章为绪论，主要介绍课题背景、研究意义、国内外研究现状、本文研究内容和论文组织结构。

第 2 章为相关技术与理论，主要介绍游戏化学习、动态难度调整、程序化地图生成、Unity、Spring Boot、MySQL 和 RESTful 接口等内容。

第 3 章为需求分析，主要对系统用户角色、功能需求、非功能需求和核心用例进行说明。

第 4 章为系统总体设计，重点介绍系统总体架构、模块交互流程、学力值更新模型、难度映射与地图交互设计，以及节点式地图生成算法。

第 5 章为系统实现，结合真实项目说明 Unity 玩家端、Spring Boot 后端、MySQL

数据库、学力值更新算法、题目战斗统计、成就回传和 Vue 管理后台等模块的实现方式。

第 6 章为系统测试，主要介绍测试环境、系统主要功能测试、学生端和管理端运行结果展示，并对测试结果进行分析，以验证系统主要功能和数据闭环的可行性。

第 7 章为结论，主要总结本文完成的主要工作、系统实现效果以及当前系统仍存在的不足，并提出后续可以继续改进和完善的方向。

第2章 相关技术与理论

2.1 系统技术路线概述

本文系统面向教育游戏中的学习行为反馈和地图自适应问题，整体技术路线可以概括为“学习行为采集—学力值计算—地图参数映射—节点地图生成—题目战斗交互—数据回传更新”。本系统不是只为了做出游戏画面效果，而是想把学习行为放进游戏流程中，让学习数据能影响地图结构、关卡难度和奖励反馈。

从功能实现来看，系统大体分为三层。第一层是 Unity 客户端表现层，主要负责节点地图显示、地图节点生成、进入关卡、题目作答、小车战斗、奖励弹窗和成就树展示。第二层是 Spring Boot 后端服务层，主要处理用户认证、学习行为记录、学力值计算、难度配置、题目查询、战斗记录接收和成就同步。第三层是 MySQL 数据存储层，用来保存用户、题目、难度配置、游戏结果、答题记录、复习会话、奖励和成就等数据。

从理论基础来看，游戏化学习说明了为什么可以把目标、反馈和奖励加入学习过程；动态难度调整说明了为什么要根据学习者表现改变挑战强度；程序化地图生成则给节点地图自动生成提供了方法参考^{[1][6][10]}。这些理论内容和 Unity、Spring Boot、MySQL、RESTful 接口等工程技术一起，构成了本文系统设计和实现的基础。

2.2 游戏化学习

游戏化学习是指将游戏中的目标设定、关卡挑战、即时反馈、积分奖励、角色成长和成就系统等机制引入学习活动，以提升学习者的学习动机和持续参与度。相关综述表明，游戏化学习研究通常关注学习特征、游戏元素设计、教学应用效果和学习工具评估等内容^[1]。游戏可以使玩家感受到挑战、好奇、临场感、合作、沉浸式等多种体验。基于游戏的学习已被证实是一种有效的教学方法，可以提高学生在数学学习中的动机和表现。与传统练习系统相比，游戏化学习更强调过程体验。学习者不只是完成题目，而是在任务推进和反馈循环中不断获得目标感。教育游戏的设计重点在于将学习目标、任务挑战和反馈机制整合到同一体验过程中，使游戏机制服务于知识理解和学习投入提升^[15]。

本文系统中的游戏化学习机制，主要体现在三个方面。在地图层面，系统把学习

路径转化为节点式地图，学生通过选择路线推进学习任务；在关卡层面，系统把题目作答结果转成小车战斗中的行为。学生答对题目时，小车会触发攻击反馈，答错或超时时，小车生命值会被扣减。在长期反馈层面，系统通过道具奖励和成就树保存学习进展，连续学习、能力提升等情况可以用比较直观的方式显示出来。

这种设计的重点不是单纯增加游戏外观，而是让学习行为成为游戏反馈的触发条件。已有研究综述中提到，游戏化学习常见元素包括排行榜、分数、徽章和关卡，但这些元素是否有作用，还要通过学习投入、满意度和内在动机等方面体现出来^[2]。对学生来说，答题不只是得到一个分数，还会影响地图探索、小车生存和奖励获取；对系统来说，学习行为也不只是被记录下来，还可以作为地图生成和关卡难度调整的输入。

2.3 动态难度调整

动态难度调整是指系统根据用户表现自动调整任务难度，使挑战水平与用户能力相匹配^[6]。在游戏中，动态难度调整可以通过敌人血量、敌人数量、资源分布、关卡长度、路线复杂度等方式实现。在教育游戏中，动态难度调整还需要考虑学习目标，不能只追求游戏刺激，而要使任务难度有助于知识巩固和能力提升。

本文系统将动态难度调整分为两个层次。第一层是地图层面的难度调整，主要通过路线数量、路线分支、Boss 节点数量和奖励节点数量体现。低学力用户面对更简单的地图，高学力用户面对更复杂的地图。第二层是关卡层面的难度调整，主要通过题目难度、答题时间限制体现。地图层面控制整体学习路径，关卡层面控制单次任务挑战。

动态难度调整首先要解决的问题，是如何判断用户能力。已有研究表明，动态调节挑战强度会影响用户享乐体验和成就目标取向，也可以和心流理论结合起来，使挑战状态保持在相对合适的范围内^{[7][9]}。贺松年针对游戏关卡难度动态调整机制进行了设计，强调难度调节需要结合玩家状态、关卡反馈和挑战强度变化，让关卡体验保持在较合适的区间^[16]。本文把这一思路迁移到教育游戏场景中，通过学力值把学生答题表现映射到地图复杂度和题目难度配置上。学力值由后端根据答题正确率、通关率、学习投入、习惯得分，以及错误和超时情况计算，并作为地图生成和题目选择的输入。通过这种方式，系统能把用户在过去关卡中的表现转化为下一次地图和关卡的变化。

2.4 程序化地图生成

程序化地图生成是通过算法自动生成地图结构或关卡内容的方法。程序化内容生成研究通常关注地形、地图、关卡、系统和场景等游戏内容的自动生成^[10]。常见生成对象包括地形、道路、房间、节点、敌人、资源和事件等。程序化生成能够减少人工配置成本，并增加游戏内容的多样性。对于教育游戏而言，地图生成还需要满足学习路径可控、难度可解释和节点可追踪等要求。

本文采用节点式地图生成方案。地图由起点、若干层中间节点和终点组成，节点之间通过路线连接。每个节点代表一个关卡或事件，节点类型包括普通战斗节点、Boss 节点、奖励节点和特殊节点等。系统先由后端根据学力值返回难度档位，再由 Unity 客户端选择对应的 MapConfigSO 配置，在本地生成节点列表和连线关系并绘制地图。

节点式地图适合本文系统的原因在于：第一，它结构清晰，便于表达学习路径；第二，它容易控制路线数量和分支复杂度；第三，它可以通过节点类型表达题目难度、Boss 挑战和奖励反馈；第四，它便于在 Unity 端根据难度配置快速生成和展示。与完整开放世界地图相比，节点式地图更适合原型系统中的自适应难度验证。相关游戏实现研究也表明，Roguelike 类游戏常通过随机地图、战斗系统和路径组织形成可重复的关卡体验^[14]。

2.5 Unity 客户端开发技术

Unity 是常用的游戏开发引擎，支持 2D 和 3D 场景构建、UI 系统、物理系统、动画系统和 C#脚本开发。本文系统使用 Unity 实现教育游戏客户端，内容主要包括地图界面、节点交互、关卡加载、答题 UI、小车战斗、奖励弹窗和成就树界面。

在地图界面中，Unity 会根据后端返回的难度配置选择对应的地图配置资源，再通过 MapGenerator、ThreeMapGenerator 等脚本实例化地图节点，并按照本地生成的连接关系绘制节点之间的连线。不同节点类型会使用不同图标或颜色进行区分，使用户在进入关卡前就能大致判断路线风险和收益。在战斗界面中，Unity 展示题目、选项、倒计时、小车血量和敌人血量，并根据答题结果播放攻击或扣血反馈。

本文系统中的 Unity 客户端不直接承担复杂数据分析任务，而是主要负责交互表

现和地图本地生成。学力值计算、题库查询和学习行为存储由后端完成。这样的分工能够降低客户端业务耦合度，也便于后续扩展题库管理和学习数据分析功能。

2.6 Spring Boot 后端与数据服务技术

Spring Boot 是基于 Spring 框架的快速开发框架，能够通过自动配置、内嵌 Web 容器和统一项目结构降低 Java Web 项目的开发复杂度。本文系统后端采用 Spring Boot 构建数据服务，主要负责用户学习行为记录、学力值计算、难度配置维护、题目数据查询、战斗记录上传，以及奖励成就状态更新等功能。

在系统架构中，Spring Boot 后端位于 Unity 客户端和 MySQL 数据库之间，主要承担业务处理和数据组织职责。Unity 客户端不会直接访问数据库，而是通过 HTTP 接口向后端请求数据。后端接收到请求后，先由 Controller 层完成接口接收和参数校验，再由 Service 层处理学力值计算、难度档位匹配、题目筛选和结果保存等业务逻辑，最后通过数据访问层与 MySQL 数据库交互。这样的分层结构能够使接口、业务逻辑和数据操作相互分离，提高系统可维护性。

本文系统中，Spring Boot 的作用主要体现在以下几个方面。第一，提供 RESTful API，使 Unity 客户端能够获取用户状态、难度配置和题目数据。第二，集中处理学习行为数据，避免客户端直接承担复杂计算。第三，统一维护题库、学习记录、游戏结果、复习记录和成就奖励等数据。第四，为管理端、题库维护功能和学习数据统计功能提供接口基础。

2.7 MySQL 数据库与数据交互技术

MySQL 是常用的关系型数据库管理系统，具有结构清晰、查询效率较高、事务支持完善和使用成本较低等特点，适合中小型 Web 系统和原型系统的数据存储。本文系统采用 MySQL 保存用户信息、题目数据、难度配置、游戏结果、题目作答记录、复习会话、奖励信息和成就进度等数据。

教育游戏系统中的数据有比较明显的结构化特点。在本系统中，一个用户可以产生多条游戏结果记录，一条游戏结果又可以对应多条题目作答记录，同一个用户还可能有多条成就进度和复习会话记录。这样的数据关系比较适合放在关系型数据库中处理。系统通过主键和类似外键的关联方式，可以追踪某个用户的学习行为来源，也可

以查看某次关卡战斗里题目表现和学力值变化。

Unity 客户端和 Spring Boot 后端之间通过 RESTful 接口和 JSON 格式交换数据。难度配置一般包含 difficultyLevel、studyAbilityMin、studyAbilityMax 和 description；题目数据包含题目编号、题干、选项、正确答案和难度；战斗记录则包含用户编号、答题结果、耗时、胜负结果、奖励情况和题目作答明细。这种方式结构比较清楚，调试时也比较方便，后续可以用接口测试工具检查后端服务是否正常。

2.8 本章小结

本章主要对教育游戏地图交互引擎的技术基础进行说明。游戏化学习为本文把答题、战斗、奖励和成就结合起来提供了理论依据；动态难度调整为学力值驱动地图复杂度变化提供了设计思路；程序化地图生成则为节点式地图自动生成提供了方法支撑。在工程技术方面,Unity 负责三维游戏环境下的地图表现、节点交互和题目战斗,Spring Boot 负责用户、题库、难度配置、学力值和学习结果等数据服务,MySQL 则负责保存结构化过程数据。上述技术共同构成了后续系统总体设计和功能实现的基础。

第3章 需求分析

3.1 可行性分析

3.1.1 技术可行性分析

从项目目前的实现情况来看，系统已经形成了前后端分层的基本结构。后端使用 Spring Boot 提供统一接口，已经包含用户认证、游戏配置、题目获取、游戏结果保存、学力值更新、成就同步、题目统计、复习会话和后台管理等功能。管理前端使用 Vue3 实现，能够进行题库管理、难度配置、玩家数据管理和统计看板展示。Unity 玩家端主要完成登录注册、地图展示、题目战斗和学习数据采集。前后端之间通过请求和响应传递数据，整体技术路线也符合 Web 系统和游戏客户端联调的常见方式。

这种结构的优点是客户端和后端职责比较清楚，功能边界也容易划分。玩家端主要通过登录注册接口、游戏配置接口、用户数据接口、成就同步接口、题目获取接口和复习训练接口完成交互；管理端则通过后台接口完成题库维护、难度配置、玩家数据查看和统计分析。从现有控制器、数据模型和页面功能来看，系统主要业务链路已经能够连起来，可以支撑玩家从登录、进入地图、完成答题战斗，到保存成绩、更新学习数据、生成复习内容的完整流程。因此，从工程实现角度看，本课题具备技术可行性。

地图生成部分采用节点式拓扑结构，在三维场景中展示节点、路线和关卡入口，没有直接生成复杂三维地形。相关程序化内容生成研究说明，游戏内容可以在规则限制下自动生成，节点、关卡和路径结构也可以作为地图生成对象^{[10][13][14]}。这种方案能在学习路径比较可控的情况下，实现路线分支、Boss 分布和奖励反馈等效果，也为后续路径动画、角色移动和场景表现优化留下空间。结合项目实现来看，地图节点和连线由 Unity 客户端根据 MapConfigSO 配置在本地生成，后端主要维护学力值、难度区间、题库和学习结果数据。对本科毕业设计来说，这种实现方式同时包含三维地图交互、地图生成算法、前后端数据交互和游戏化学习流程，能够满足设计与实现类课题的要求。

3.1.2 经济可行性分析

本系统开发主要使用 Unity、Spring Boot、Vue3 和 MySQL 等常见工具，这些工

具比较成熟，使用成本也不高，不需要另外购买专门硬件。系统中的题库、难度配置、玩家成绩、学力值数据和复习记录等内容，可以存放在本地数据库或普通服务器中，所以开发和部署成本相对可控。

从原型系统角度看，玩家端、管理端和后端服务都可以在普通个人电脑上开发和调试。后续联调时，只需要搭建简单服务器环境，或者先在本机完成测试。管理端提供题库和难度配置功能以后，系统内容更新不必频繁修改客户端程序，这也能减少后期维护工作量。因此，系统在开发成本、部署成本和维护成本方面具有较好的经济可行性。

3.1.3 操作可行性分析

系统操作流程符合常见教育游戏和后台管理系统的使用习惯。玩家端主要围绕登录注册、选择难度、进入地图、答题战斗、查看奖励和复习错题展开，整体流程比较直观，用户不需要理解复杂的后端逻辑也能使用。地图以节点和路线的形式呈现学习路径，能够降低复杂场景带来的操作负担，使学生更容易理解当前关卡位置、后续路线和挑战目标。

管理端采用表格、筛选、分页、弹窗和统计图表等常见交互方式，比较适合教师或管理员进行题库管理、难度管理、玩家数据查看和学习情况分析。后台操作主要围绕数据维护和统计查看展开，功能边界清楚，学习成本也不高。因此，从用户操作习惯和界面交互方式来看，本系统具有较好的操作可行性。

3.2 需求分析

3.2.1 系统需求概述

根据课题原始要求，系统需要围绕教育游戏中的地图交互、学习行为反馈和习惯养成激励来展开设计。游戏化学习研究强调目标、反馈、奖励和学习动机之间的关系，动态难度调整研究则强调根据用户表现调整后续挑战强度^{[1][2][6]}。因此，本系统的主要目标不是做一个普通小游戏，而是把学习行为数据、地图生成、关卡挑战、奖励反馈和行为回传连接起来，让地图成为学习进度和游戏机制之间的中间部分。

结合项目实现情况，本文把课题需求落实为一个以三维节点式拓扑地图为核心的教育游戏地图交互引擎。系统根据学生的学习时长、答题正确率、错题情况、当前难度、通关结果和连续学习情况计算学力值，再根据学力值生成复杂度不同的关卡地图。

地图中的普通节点对应常规题目战斗，Boss 节点对应难度较高的挑战，奖励节点对应阶段性奖励，这些节点和路线都会在三维场景中展示和交互。

学生进入系统后，Unity 会按照当前学力值对应的难度配置生成地图。在关卡中，学生通过答题控制小车与敌人战斗，答题结果会影响攻击、扣血和结算。关卡结束后，系统记录答题结果、耗时、错题数量、胜负结果和奖励情况，并把这些数据传回后端，作为后续学力值更新和难度匹配的依据。

需要说明的是，长期教育效果验证和更复杂的三维场景细节优化，属于后续可以继续开展的工作。本文原型系统主要完成了与课题目标对应的三维地图交互、题目战斗、数据回传和可运行验证。

3.2.2 用户角色分析

系统主要涉及学生玩家和管理员两类角色。学生玩家是系统的主要使用者，需要完成登录注册、地图探索、关卡挑战、题目作答、错题复习、奖励领取和成就查看等操作。管理员则主要负责维护题库、配置难度参数、查看学习记录和调整奖励规则。

学生玩家的核心需求，是获得清晰、有反馈、有挑战但不过度挫败的学习体验。已有心流体验和动态难度研究都强调，任务挑战需要和用户能力状态保持比较合理的匹配关系^{[5][6][7]}。教育游戏需要把知识练习嵌入明确的任务情境中，让学习活动不再只是孤立刷题，而是和关卡推进、反馈、成就感结合起来^[18]。因此，对于能力较弱的学生，系统需要提供较短路线和较少 Boss，让他们能够完成基础关卡；对于能力较强的学生，系统需要提供更复杂路线和更高挑战，使其保持学习兴趣。

管理员的核心需求，则是能够维护题目、查看数据和调整规则。当前原型系统重点实现学生端学习与游戏交互流程，管理端功能作为配套支撑，但在数据库结构和后端接口设计中，已经保留了题库维护、难度配置、玩家数据管理和统计看板等能力。

3.2.3 功能需求分析

根据课题目标和原型系统的实际实现情况，系统功能需求可以分为玩家端学习交互、后端数据处理和管理端维护分析三个层面。玩家端重点满足学生在教育游戏中的登录、地图探索、关卡答题、奖励查看和错题复习需求，并承担地图节点生成与交互表现；后端重点完成用户数据、学习行为、学力值、难度配置、题目查询和题目统计等业务处理；管理端面向教师或管理员，提供题库维护、难度配置、玩家数据查看和

统计分析等功能。各功能模块围绕“学习行为采集—学力值计算—难度配置匹配—地图生成—题目战斗—结果反馈”这一主线展开，形成较完整的学习闭环。这样的功能划分，一方面对应游戏化学习中常见的关卡、奖励和反馈机制，另一方面也对应动态难度调整中“表现数据—难度变化”的基本思想^{[2][6]}。系统功能需求如表 3-1 所示。

表 3-1 系统功能需求表

功能模块	功能内容	说明
账号认证模块	用户注册、用户登录、Token 维护	为玩家端和管理端提供身份认证
学习行为记录模块	记录学习时长、正确率、错题、当前难度、耗时和通关结果	为学力值计算提供基础
学力值计算模块	根据学习行为计算用户能力水平	将学习数据转化为地图生成输入
节点式地图生成模块	根据学力值生成不同复杂度的地图	控制路线数量、Boss 分布和奖励分布
题目战斗模块	获取题目、答题判定、攻击和扣血	将学习结果转化为即时游戏反馈
错题复习模块	创建复习会话、获取复习题目	支持针对错题的再次练习
奖励与成就模块	根据关卡表现和连续学习行为发放奖励、更新成就	提供短期激励和习惯养成反馈
题目统计模块	上报题目作答情况、统计正确率	支持题库分析和题目优化
后台管理模块	题库管理、难度管理、玩家管理、游戏结果管理、统计看板	为管理员提供维护和分析能力

从原始课题需求与实际实现的对应关系看，本文系统对需求进行了原型化落地，如表 3-2 所示。

表 3-2 原始需求与本文实现对应表

原始需求方向	本文实现方式	实现说明
学习行为驱动地图动态生成	节点式拓扑地图动态生成	以路线、层数、Boss 节点和奖励节点体现地图复杂度
学习行为数据采集与反馈	学习时长、正确率、错题、当前难度等记录	为学力值计算与关卡调整提供输入
习惯养成量化与反馈	奖励与成就模块、错题复习模块	将连续学习、正确率提升和错题练习转化为反馈
原型系统集成与验证	Unity+SpringBoot+Vue+MySQL 原型系统	通过功能测试、地图对比测试、接口测试和数据闭环验证可行性

3.2.4 非功能需求分析

除功能需求外，系统还需要满足性能、安全、数据一致性、可维护性、可扩展性和易用性等非功能需求。由于本文系统同时包含 Unity 玩家端、Spring Boot 后端、

MySQL 数据库和 Vue 管理后台，非功能需求不仅关系到界面能否正常操作，也关系到学习行为数据能否被稳定记录、正确计算和持续复用。

系统非功能需求如表 3-3 所示。

表 3-3 系统非功能需求表

需求类别	具体要求	设计与实现依据
性能响应需求	登录、题目获取、游戏结果上传和统计查询应在较短时间内完成响应，避免影响游戏连续性	后端按接口拆分业务职责，常用查询字段建立索引，题目、游戏记录和统计数据按难度、用户和时间组织
数据安全需求	玩家和管理员账号需要区分管理，密码不得明文保存，接口访问需要具备身份校验能力	系统使用 Token 维护登录状态，用户密码以 password_hash 字段保存，管理员账号单独存放在 admin_users 表
数据一致性需求	学力值、题目难度、游戏结果、做题明细和成就进度需要保持可追踪关系	数据库通过外键关联 users、game_results、user_question_attempts、review_sessions 和 user_achievements，并通过 CHECK 约束限制 study_ability、difficulty 和 status 等字段范围
可维护性需求	题库、难度区间和统计规则应便于后台维护，减少修改客户端程序的频率	difficulty_config 表保存学力值区间与难度档位，Vue 管理后台提供题库、难度配置、玩家数据和统计看板维护入口

在性能方面，玩家端的主要交互集中在进入地图、加载题目、完成答题和上传结果几个节点。如果题目加载或结果上传时间过长，用户会很明显地感受到游戏流程被打断。因此，系统把题库查询、游戏结果保存、题目统计更新等操作拆分为独立接口，并在数据库中为 user_id、difficulty、created_at、status 等字段建立索引，使后台分页查询、玩家时间线查询和题目统计查询有较稳定的响应基础。

在安全和数据一致性方面，系统把普通玩家账号和管理员账号分开保存，这样可以减少管理端权限和玩家端数据混用的情况。用户状态由 status 字段控制，如果账号异常或被禁用，就不能继续作为正常玩家参与系统流程。学习力和难度字段也被限制在合理范围内，题目作答记录通过 game_result_id 或 review_session_id 关联到对应来源，这样一次答题行为可以追溯到普通游戏流程，也可以追溯到错题复习流程。

在可维护性和扩展性方面，系统没有把学习力区间和难度档位固定写在 Unity 客户端中，而是交给 difficulty_config 表和后台难度配置页面维护。后续如果需要调整难度策略，可以优先修改后端配置。user_question_attempts 表记录单题作答明细，question_stats 表记录题目尝试次数和正确次数，这些数据可以为后面的题目质量分析、

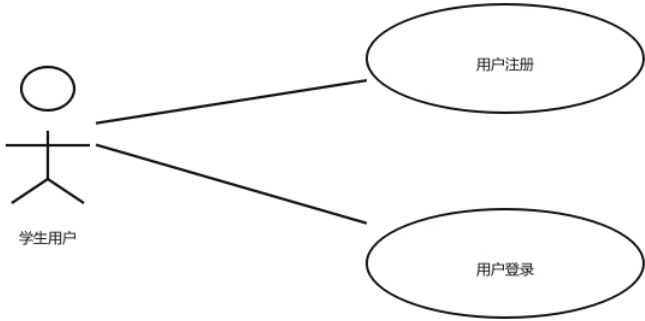
错题复习和个性化推荐提供基础。

3.2.5 核心用例分析

系统分为学生端和管理端两部分，学生端功能拆分为账号管理、学习地图、关卡答题、错题复习四个小模块。

账号管理模块主要包含用户注册和用户登录两个用例。该模块用于完成学生用户身份建立和身份验证，是学生端其他功能的基础。如表 3-4 所示。

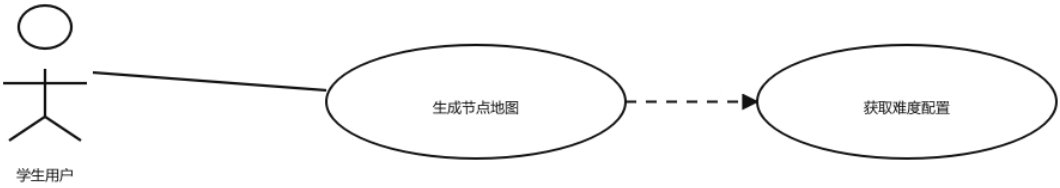
表 3-4 账号管理模块

用例名称	账号模块用例	参与者	学生玩家
			
主执行者	学生玩家		
功能描述	学生用户通过注册功能创建个人账号，并在后续使用系统时通过登录功能完成身份验证。账号管理模块为节点地图、关卡答题和错题复习等功能提供用户身份基础。		
前置条件	学生用户能够访问系统；系统注册和登录服务正常运行。		
后置条件	注册成功后，系统保存学生用户信息；登录成功后，系统建立学生用户的登录状态，并允许其访问学生端功能。		
主成功场景	1. 学生用户进入账号管理页面。 2. 若用户尚未拥有账号，则填写注册信息并提交。 3. 系统校验注册信息，校验通过后创建账号。 4. 学生用户输入账号和密码进行登录。 5. 系统验证账号密码，验证通过后进入学生端主页。		
扩展	1. 若注册账号已存在，系统提示用户更换账号。 2. 若注册信息不完整或格式错误，系统提示用户修改。 3. 若登录账号或密码错误，系统提示用户重新输入。 4. 若服务异常，系统提示用户稍后重试。		
包含	用户注册、用户登录。		
业务规则	学生账号应保持唯一；密码应满足系统设置的安全要求；未登录用户不能访		

	问关卡答题、错题复习等需要身份认证的功能。
--	-----------------------

学习地图模块主要包含生成节点地图和获取难度配置两个用例。其中生成节点地图包含获取难度配置，用于根据学生学习进度生成可视化学习路径。具体说明如表 3-5 所示。

表 3-5 学习地图模块

用例名称	学习地图模块用例	参与者	学生玩家
			
主执行者	学生玩家		
功能描述	学生用户登录后进入学习地图页面，系统根据学生当前学习进度和难度配置生成节点地图，展示已完成、可挑战和未解锁的学习节点。		
前置条件	学生用户已成功登录；系统存在学生学习进度数据和关卡难度配置。		
后置条件	系统生成并展示学生当前可用的节点地图，学生可选择已解锁节点进入后续关卡答题。		
主成功场景	1. 学生用户进入学习地图页面。 2. 系统读取学生当前学习进度。 3. 系统获取对应的难度配置。 4. 系统根据难度配置生成节点地图。 5. 系统展示节点状态，学生选择可挑战节点。		
扩展	1. 若难度配置不存在，系统使用默认配置生成节点地图。 2. 若节点尚未解锁，系统提示学生需要先完成前置节点。		
包含	生成节点地图、获取难度配置。		
业务规则	难度配置应与学力值数值所在阶段相匹配。		

关卡答题模块是学生端核心模块，主要包含关卡答题战斗、获取题库、记录学习行为、保存游戏结果、更新学力值、同步成就和上报题目统计等用例。在小模块图中，可将关卡答题战斗作为中心用例，其他用例作为其包含或后续处理流程。具体说明如表 3-6 所示。

表 3-6 关卡答题模块

用例名称	关卡答题模块用例	参与者	学生玩家
<pre> graph LR User((学生用户)) --- Battle((关卡答题战斗)) Battle -.-> GetBank((获取题库)) Battle -.-> RecordBehav((记录学习行为)) Battle -.-> SaveResults((保存游戏结果)) Battle -.-> ReportStats((上报题目统计)) Battle -.-> UpdateValue((更新学力值)) SaveResults -.-> SyncAch((同步成就)) SaveResults -.-> UpdateValue </pre>			
主执行者	学生玩家		
功能描述	学生用户选择学习地图中的可挑战节点后进入关卡答题流程。系统根据关卡要求获取题目，学生完成答题后，系统记录学习行为、保存游戏结果，并根据结果更新学力值、同步成就和上报题目统计。		
前置条件	学生用户已登录；学生已选择可挑战节点；系统题库、学习记录和结果保存服务正常运行。		
后置条件	系统完成本次关卡结算，保存学生答题结果和学习行为，并更新相关学习数据。		
主成功场景	1. 学生用户在节点地图中选择可挑战关卡。 2. 系统根据学力值数值和知识点要求获取题库。 3. 学生用户进入关卡答题战斗并提交答案。 4. 系统判断答案正确性并记录学习行为。 5. 关卡结束后，系统保存游戏结果。 6. 系统根据答题表现更新学力值。 7. 系统同步成就状态并上报题目统计。 8. 系统向学生展示关卡结算结果。		
扩展	1. 若题库获取失败，系统使用本地题库兜底处理。 2. 若学生中途退出，系统根据规则保存已完成的答题记录。 3. 若结果保存失败，系统提示稍后同步或重新提交。 4. 若成就同步或统计上报失败，系统可在后台进行补偿处理。		
包含	关卡答题战斗、获取题库、记录学习行为、保存游戏结果、更新学力值、同步成就、上报题目统计。		
业务规则	关卡结果应根据学生实际答题情况计算；学习行为记录应与学生、题目和关卡关联；同一次关卡结果不能重复结算；学力值和成就变化应与答题表现一		

	致。
--	----

错题复习模块主要包含错题复习、获取复习题目和创建复习会话三个用例。其中错题复习作为中心用例，包含获取题目和创建会话两个辅助流程。具体说明如表 3-7 所示。

表 3-7 错题复习模块

用例名称	错题复习模块用例	参与者	学生玩家
<pre> graph LR User((学生用户)) --- Review([错题复习]) Review -.-> Get([获取复习题目]) Review -.-> Create([创建复习会话]) </pre>			
主执行者	学生玩家		
功能描述	学生用户进入错题复习模块后，系统根据历史错题记录获取复习题目，并创建本次复习会话。学生通过重新作答错题巩固薄弱知识点，系统记录复习结果。		
前置条件	学生用户已登录；系统中存在该学生的历史答题记录；错题数据可正常查询。		
后置条件	系统创建复习会话并保存本次复习过程，学生可查看复习结果或继续进行后续复习。		
主成功场景	1. 学生用户进入错题复习页面。 2. 系统查询学生历史错题记录。 3. 系统筛选并返回复习题目。 4. 系统创建复习会话。 5. 学生用户完成错题作答。 6. 系统反馈答题结果并记录复习情况。 7. 系统展示本次复习结果。		
扩展	1. 若学生暂无错题，系统提示暂无可复习内容。 2. 若复习题目获取失败，系统提示用户稍后重试。 3. 若学生中途退出复习，系统保存已完成的复习进度。		
包含	错题复习、获取复习题目、创建复习会话。		
业务规则	复习题目应来自学生个人历史错题记录；错题复习应优先覆盖错误次数较多或近期未掌握的知识点；复习记录应与学生和复习会话关联。		

管理端功能主要面向系统管理员，用于完成后台数据维护、题库配置、学习数据

统计和结果管理等工作。与学生端相比，管理端用例之间的包含关系较少，整体结构较为清晰，因此管理端可采用一张总用例图进行展示，无需继续拆分为多个小模块用例图。具体说明如表 3-8 所示。

表 3-8 管理端模块

用例名称	管理端用例	参与者	管理员
<pre> graph LR Admin((管理员)) --> Login([管理员登录]) Admin --> PlayerData([玩家数据管理]) Admin --> QBank([题库管理]) Admin --> Config([难度配置管理]) Admin --> Stats([题目作答统计]) Admin --> Results([游戏结果管理]) Admin --> Dashboard([统计看板]) Admin --> Review([复习会话管理]) </pre>			
主执行者	管理员		
功能描述	管理员登录后台管理系统后，可对玩家数据、题库内容、难度配置、题目作答统计、游戏结果、统计看板和复习会话进行管理，为学生端学习过程和教育游戏运行提供后台支持。		
前置条件	管理员具有合法后台账号；后台管理系统和相关数据服务正常运行。		
后置条件	管理员完成对应后台操作后，系统保存管理结果或展示统计数据，相关配置和数据可被学生端或统计模块继续使用。		
主成功场景	1. 管理员进入后台登录页面。 2. 管理员输入账号和密码并提交。 3. 系统验证管理员身份，验证通过后进入后台首页。 4. 管理员根据需要选择玩家数据管理、题库管理、难度配置管理、题目作答统计、游戏结果管理、统计看板或复习会话管理功能。		

	5. 系统展示对应功能页面。 6. 管理员进行查询、新增、修改、删除或查看统计结果等操作。 7. 系统保存操作结果或返回查询统计结果。
扩展	1. 若管理员账号或密码错误，系统提示登录失败并要求重新输入。 2. 若数据查询失败，系统提示当前数据暂不可用。 3. 若配置保存失败，系统提示管理员检查输入内容或稍后重试。
包含	管理员登录、玩家数据管理、题库管理、难度配置管理、题目作答统计、游戏结果管理、统计看板、复习会话管理。
业务规则	管理员必须登录后才能访问后台功能；题库和难度配置的修改应保证学生端关卡答题流程可正常使用；统计数据应来源于真实的学生答题记录和游戏结果；关键数据修改应进行必要的权限控制。

3.3 本章小结

从可行性来看，系统采用 Unity、Spring Boot、Vue3 和 MySQL 等较成熟的技术，可以支撑玩家端游戏交互、后端数据服务和管理端维护分析，开发成本和操作难度也在可以接受的范围内。从需求来看，系统围绕学生玩家和管理员两类角色展开，需要完成登录注册、节点式地图生成、题目战斗、学力值更新、错题复习、奖励成就和后台管理等功能。

结合项目实现情况，本文把完整三维地图动态生成需求做成了原型系统。Unity 玩家端负责节点地图生成和战斗表现，Spring Boot 后端负责难度配置、题库、学习结果和学力值更新，Vue 管理后台负责内容维护和数据查看。本章的需求分析，为后续系统总体设计、数据库设计和功能实现提供了依据。

第4章 系统总体设计

4.1 系统总体设计

本文系统围绕“学习行为驱动地图交互”的目标进行设计，整体由 Unity 玩家端、Spring Boot 后端、MySQL 数据库和 Vue 管理后台组成。Unity 玩家端负责地图节点展示、题目战斗、奖励反馈和成就表现；Spring Boot 后端负责认证、难度配置、题库查询、游戏结果保存、学力值更新、成就同步和题目统计；MySQL 数据库负责保存用户、题目、难度配置、游戏结果、成就和统计数据；Vue 管理后台负责题库维护、难度配置、玩家管理和数据分析。

图 4-1 展示了系统总体架构。该图采用分层方式说明各部分职责：玩家端不直接访问数据库，而是通过后端接口完成数据交互；管理后台也通过后端管理接口维护题库和难度配置；数据库则作为统一的数据持久化层，为玩家端流程和后台统计提供数据基础。

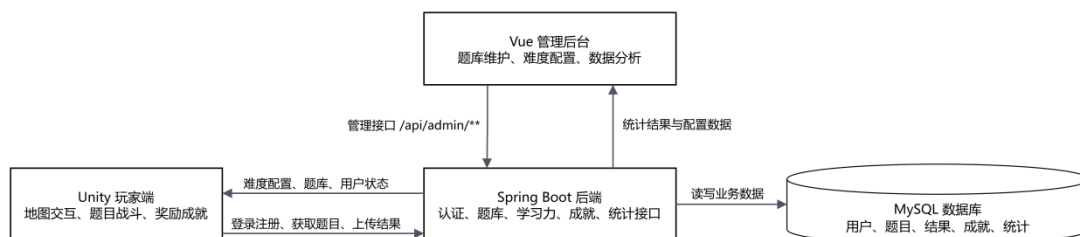


图 4-1 系统总体架构

4.2 模块交互流程设计

本文把模块交互抽象为“预输入数据—难度评估—地图与任务生成—用户交互—学习训练—任务点判定—进度更新或错题复习”的闭环。该闭环借鉴了动态难度调整中依据玩家表现调节后续挑战的思想，也结合了游戏化学习中即时反馈和阶段奖励的机制^{[2][6][7]}。用户状态、历史学习结果、题目统计和历史错题会作为预输入数据进入难度评估环节；系统根据掌握程度提升或降低难度，并进一步生成巡线路径和任务点；Unity 玩家端完成地图渲染与用户交互后，玩家进入学习训练。若未通过关卡，则重新进入关卡并适当降低难度；若通过关卡，则进入任务点判定，普通任务完成后更新进度并进入下一关，全部任务完成后再根据历史错题判断是否生成复习场景。

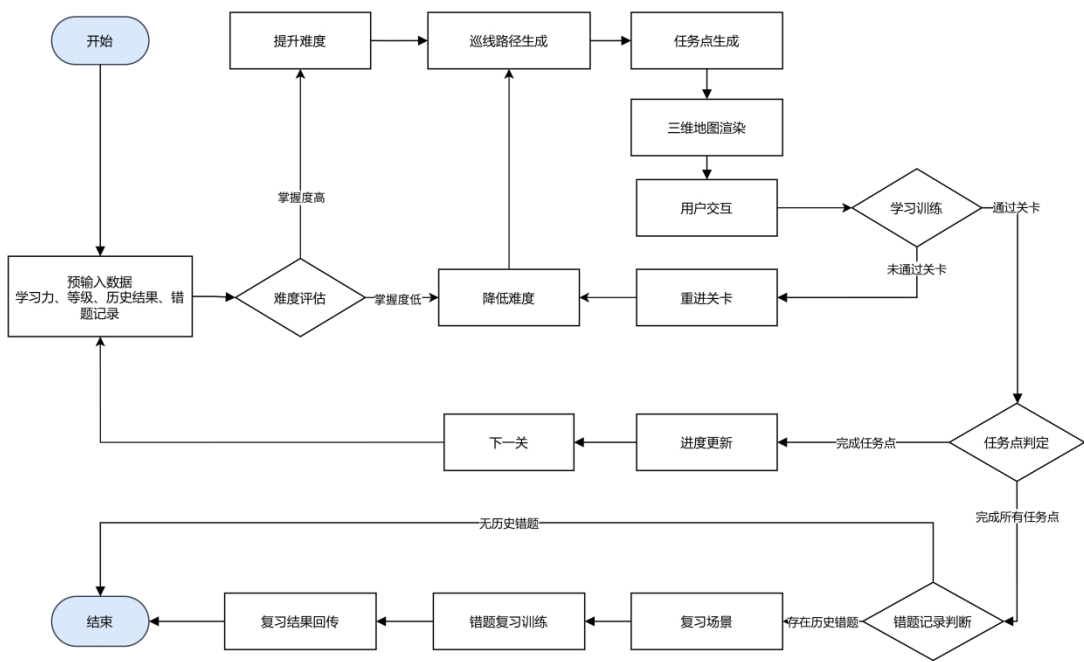


图 4-2 模块交互流程图

4.3 学力值更新模型设计

系统使用 studyAbility 表示玩家学力值。学力值取值范围为 1-100，数值越高，说明玩家近期学习表现越稳定，系统可以提供更高难度的题目或更复杂的地图节点；数值越低，说明玩家更需要明确的路径引导和相对温和的挑战强度。该设计借鉴了动态难度调整思想，也就是根据用户近期表现调节后续挑战强度，使任务难度和用户能力状态保持比较合理的匹配关系^{[6][7]}。

表 4-1 学习行为数据来源表

数据类别	关键字段	数据含义	设计作用
用户状态	studyAbility、currentLevel、experience	当前学力值、等级和经验	作为难度选择和成长反馈依据
游戏结果	playTime、roomsCompleted、isCompleted	本局耗时、完成房间数和通关状态	衡量任务完成度和游戏投入
答题表现	correctAnswers、wrongAnswers	正确题数和错误题数	衡量知识掌握情况
收集表现	goldCollected	金币收集数量	衡量地图探索和奖励完成情况
题目统计	questionId、isCorrect、difficulty	单题作答结果与题目难度	支撑题库质量分析
成就进度	achievementId、currentValue、isUnlocked	成就当前值和解锁状态	支撑长期学习反馈

当本局游戏没有通关时，系统不会只根据耗时或正确率给出普通评分，而是会把失败状态也放进计算中。具体来说，未通关局的时间效率得分最高只能到 0.30，避免玩家因为耗时短而得到过高的时间分。同时，系统会根据房间完成度计算未完成惩罚，推进房间越少，惩罚就越明显。这样处理以后，学力值下降不只来自答题错误，也能体现关卡失败和任务完成不足的情况。

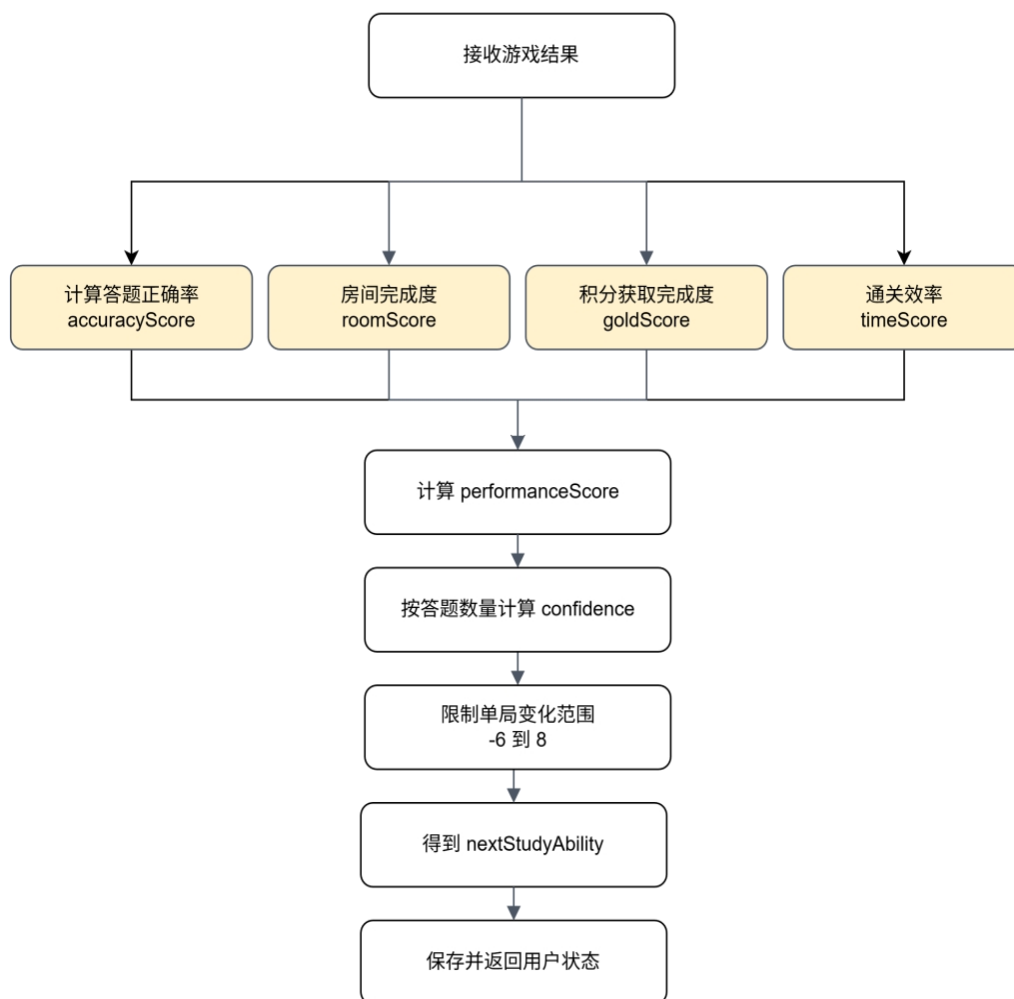


图 4-3 学力值更新流程图

学力值计算模型主要流程如图 4-3 所示。当答题正确率较高、房间完成数量较多、收集表现较好，并且耗时处于合理范围时，学力值会提高；如果表现相反，学力值则会下降。系统同时引入 confidence 和单局变化范围限制，避免少量题目或单次异常表现导致学力值出现剧烈波动。动态难度调整强调“根据过程表现调节难度，但需要避免难度突变破坏体验”的思想^{[6][7][9]}，本文的处理方式与这一思路基本一致。

4.4 难度映射与地图交互设计

难度配置由后端 `difficulty_config` 维护，Unity 游戏开始时通过 `/api/game/difficulty` 获取 `difficultyLevel`、`studyAbilityMin`、`studyAbilityMax` 和 `description` 等字段。论文中不将难度区间写死为固定常量，而是以后端返回配置为准。这样管理员可以在管理后台调整学力值区间，系统也可以在不修改 Unity 客户端逻辑的情况下改变难度策略。教育游戏中的难度自适应研究表明，固定关卡配置容易忽视学习者个体差异，而结合学习或通关表现进行难度适配有助于改善学习兴趣和参与表现^[8]。具体字段说明如表 4-2 所示。

表 4-2 难度配置字段表

字段	含义	设计作用
<code>difficultyLevel</code>	难度档位	Unity 请求题库和设置节点难度时使用
<code>studyAbilityMin</code>	学力值下限	判断该难度适用的最低学力值
<code>studyAbilityMax</code>	学力值上限	判断该难度适用的最高学力值
<code>description</code>	难度说明	用于后台维护和调试说明

4.5 节点式地图生成算法设计

节点式地图生成属于 Unity 客户端本地逻辑，后端不直接生成地图节点和连线。系统先根据用户 `studyAbility` 在 1-100 范围内匹配后端 `difficulty_config`，得到当前 `difficultyLevel`；Unity 再根据该难度档位选择对应的地图配置资源，并在本地完成房间列结构、节点类型、路径连接和空间位置的生成。该设计使地图表现具有即时性，同时保留后端难度配置的可维护性。程序化内容生成研究指出，游戏地图和关卡可以通过算法生成，并在约束条件下提供多样化内容^{[10][13]}。

表 4-3 地图生成关键参数表

参数	含义	设计作用
<code>difficultyLevel</code>	后端返回的难度档位	选择 Unity 地图配置和题目难度
<code>roomBlueprints</code>	地图列配置集合	定义每一列的节点数量范围和房间类型约束
<code>min、max</code>	单列节点数量下限与上限	控制地图规模和路线复杂度
<code>roomType</code>	房间类型	决定普通战斗、精英战斗、Boss 或休息奖励节点
<code>columnGroups</code>	按列保存的节点集合	支撑路径连接和可达节点判断
<code>allConnections</code>	节点邻接关系	保存节点之间的连线，支持地图遍历和点击限制

地图配置选择采用后端难度档位+Unity 配置资源的方式实现。`difficulty_config`

用来说明某个学力值区间对应的 difficultyLevel，Unity 端则给不同 difficultyLevel 准备不同的 MapConfigSO。低难度配置会减少列数、分支和强度较高的节点，高难度配置会增加路线选择、Boss 节点和奖励节点。这样地图复杂度可以随着学习状态变化，而不是完全依靠客户端中固定写死的阈值。PCG 相关研究也提到，内容生成的作用不只是增加内容数量，还要让生成内容受到设计目标和用户状态的限制^[12]。

节点列结构按照配置逐列生成。每一列先从 roomBlueprints 读取节点数量范围，再在 min 和 max 之间随机取得本列节点数量，随后给每个节点计算空间位置，并加入 columnGroups 中。

房间类型分配时，系统把确定规则和随机规则结合起来。第一列一般设置为普通房间，让玩家进入地图后先完成基础战斗；最后一列设置为 Boss 房间，给本轮地图提供明确终点；中间列则在普通房间、精英房间和休息房间之间，根据配置或概率生成。这样地图既有开始和结束，也能通过中间节点变化增加重复游玩的差异。

表 4-4 房间类型与题目难度映射表

房间类型	项目标识	题目难度	交互含义
普通房间	MinorEnemy	难度 1	基础战斗，适合概念巩固
精英房间	EliteEnemy	难度 2	进阶战斗，适合综合应用
Boss 房间	Boss	难度 3	关键挑战，作为地图阶段目标
休息房间	RestRoom	无题目	恢复或奖励节点，用于调节节奏

路径连接规则主要用于保证地图可达性。系统通过 allConnections 保存节点邻接关系，并在相邻列之间建立连线。生成连线时需要满足两个条件：一是从起始列到终点列至少有一条完整路径，避免出现孤立节点；二是控制每个节点的分支数量，避免路线过密导致界面不易阅读。相关 PCG 研究也强调，程序化生成并不等于完全随机，而是要在设计目标和规则约束下生成可用内容^{[10][12][13]}。玩家实际点击节点时，Unity 会根据当前逻辑位置和 allConnections 判断目标节点是否可达，从而保证学习路径按顺序推进。

空间布局采用按列展开的规则，使地图在画面中能够呈现出比较清晰的层次。设第 i 个节点的横向坐标为 x_i ，起始横坐标为 x_{start} ，节点间距为 $d_{horizontal}$ ，则横向位置可表示为：

$$x_i = x_{start} + i \times d_{horizontal} \quad (\text{公式 4.1})$$

设第 $column$ 列的纵深坐标为 z_{column} ，列间距为 d_{depth} ，则纵向位置可表示为：

$$z_{column} = column \times d_{depth} \quad (\text{公式 4.2})$$

通过上述设计，系统能够将学力值、难度配置、节点类型和题目难度统一到一套地图生成逻辑中。后端负责维护难度档位、题库和学习数据，Unity 负责将这些数据转化为可交互的节点地图，二者共同形成学习行为驱动的地图交互机制。

4.6 本章小结

本章主要对教育游戏地图交互引擎系统的总体设计进行了说明。首先，从系统整体架构出发，明确了 Unity 玩家端、Spring Boot 后端、MySQL 数据库和 Vue 管理后台之间的职责分工，说明了系统通过前后端分离和数据持久化方式完成学习行为采集、难度配置、地图生成和结果回传等核心流程。其次，围绕模块交互流程，设计了学生从登录、获取难度配置、生成节点地图、进入关卡答题，到上传游戏结果和更新学力值的完整业务闭环。

在核心设计方面，本章重点说明了学力值更新模型、难度映射与地图交互设计，以及节点式地图生成算法。学力值更新模型用于将学生答题正确率、关卡完成情况、耗时和奖励等学习行为转化为可量化的能力状态；难度映射机制则根据学力值匹配不同难度档位，并进一步影响题目难度、地图路线数量、Boss 节点和奖励节点分布；节点式地图生成算法通过分层节点、连线关系和节点类型控制，使系统能够在保持学习路径清晰的同时，生成具有一定差异性的关卡地图。

通过本章设计，系统形成了“学习行为数据采集—学力值计算—难度映射—节点地图生成—题目战斗—结果回传”的总体逻辑，为第五章的具体系统实现提供了结构依据和算法基础。

第 5 章 系统实现

5.1 系统实现环境与职责划分

系统实现采用前后端分离和数据持久化相结合的方式。Unity 玩家端负责玩家可见的地图节点、题目战斗、奖励弹窗和成就表现；Spring Boot 后端负责认证、题库、难度、游戏结果、学力值、等级、成就和题目统计接口；MySQL 数据库负责保存业务数据；Vue 管理后台负责系统内容维护和统计分析。各部分职责划分如表 5-1 所示。

表 5-1 系统实现职责划分表

组成部分	主要技术	实现职责
Unity 玩家端	Unity、C#、HTTP JSON	地图节点展示、题目战斗、奖励反馈、数据同步
Spring Boot 后端	Spring Boot、REST API、Token 认证	认证、题库、难度、学力值、成就和统计接口
MySQL 数据库	MySQL	保存用户、题目、结果、成就和统计数据
Vue 管理后台	Vue 3、Element Plus、ECharts	题库维护、难度配置、玩家管理、统计分析

各端通过接口协作，共同完成学习行为驱动的地图交互闭环。

5.2 Unity 玩家端实现

Unity 玩家端围绕“登录认证—难度选择—题库加载—地图节点—题目战斗—数据同步”实现主流程。登录或注册成功后，Unity 保存 token 和 UserData；进入游戏前调用/api/game/difficulty 获取难度配置，并根据 UserData 中的 studyAbility 选择 difficulty；随后调用/api/game/questions?difficulty={difficulty} 获取题目列表，用于地图节点或战斗流程。玩家端主流程如图 5-1 所示。

玩家端核心功能如表 5-2 所示。

该实现使玩家端既承担游戏交互表现，又能够把关键学习行为回传给后端。地图节点、题目战斗和奖励反馈属于 Unity 本地表现层；学力值、题库、成就和统计数据则通过后端接口保持一致。

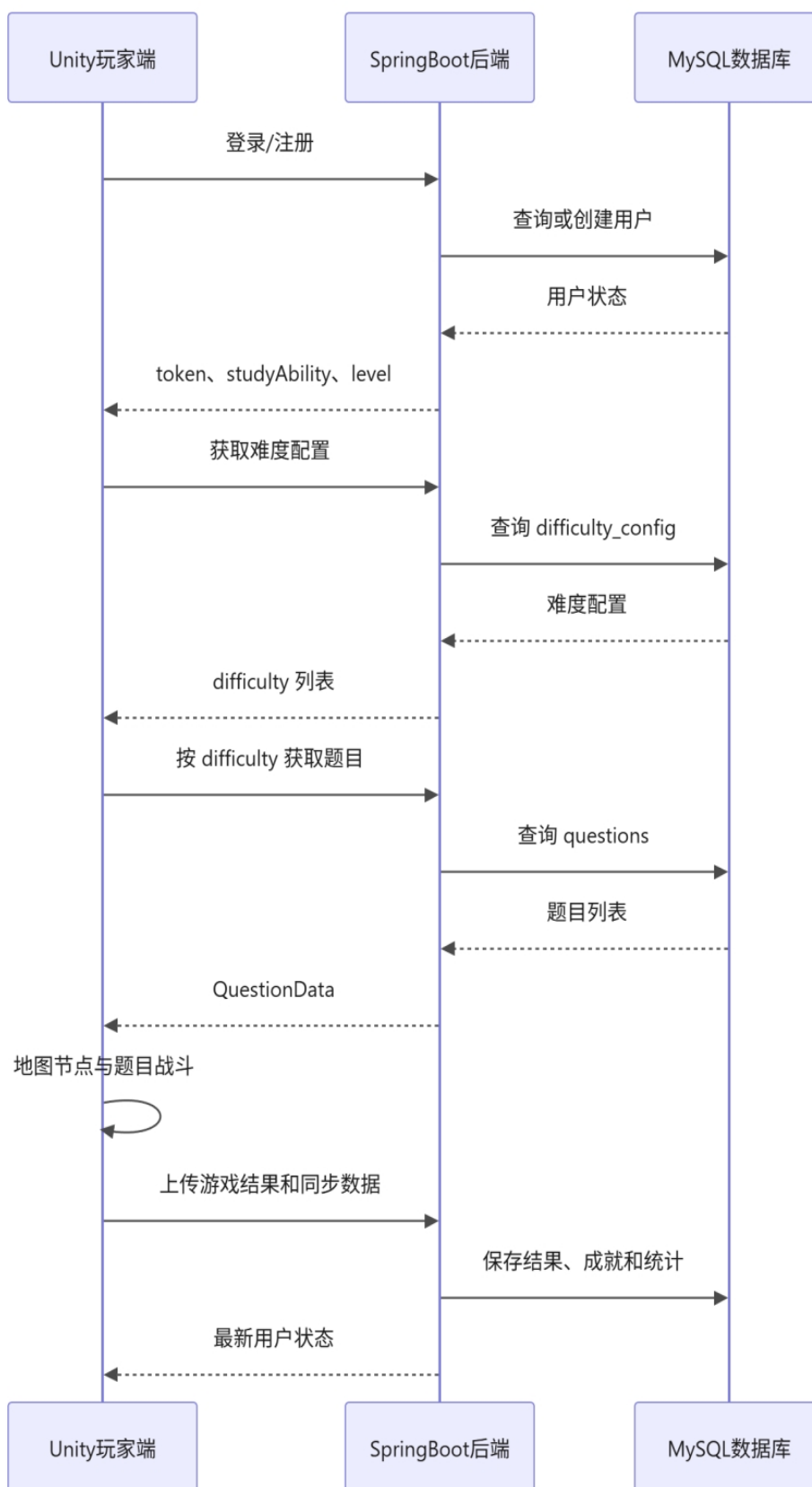


图 5-1 玩家端主流程时序图

表 5-2 Unity 玩家端功能实现表

功能模块	使用接口或数据	实现说明
登录注册	/api/auth/login、/api/auth/register	获取 token、studyAbility、等级和经验
难度选择	/api/game/difficulty	根据学力值匹配 difficulty_config
题库加载	/api/game/questions	获取对应难度题目并缓存到本局游戏
地图节点	studyAbility、difficulty、 QuestionData	根据难度组织普通、Boss 和奖励节点表现
题目战斗	QuestionData	根据答案正确性触发攻击、受击或超时反馈
游戏结算	/api/game/result	上传本局耗时、房间数、答题数、金币和通关状态，并由后端计算学力值变化
成长同步	/api/user/study-ability、 /api/user/level	同步学力值和经验，更新战斗数值
成就同步	/api/achievements、 /api/achievements/sync	读取并保存成就进度
题目统计	/api/questions/stats、 /api/questions/attempts	上报单题作答结果，支持后端统计和做题明细

5.2.1 地图配置数据实现

Unity 端使用 ScriptableObject 保存地图配置，这样不同难度档位的地图结构可以直接在编辑器中维护。地图配置的核心对象是 MapConfigSO，它内部保存多列房间蓝图，每个蓝图记录这一列节点数量范围和房间类型限制。

RoomBlueprint 用来描述单列节点的生成规则。项目中房间类型枚举写作 RoomType，正文中统一称为“房间类型”。这个命名来自已有项目代码，它表示的含义包括普通房间、精英房间、Boss 房间和休息房间。

通过这种配置方式，Unity 不需要把地图列数和节点数量直接写死在生成代码中。后端 difficulty_config 决定当前难度档位，Unity 再根据 difficultyLevel 选择对应的 MapConfigSO，从而把“学力值到难度”的数据规则，和“难度到地图表现”的客户端资源分开维护。对应 PCG 研究中“在规则和控制下生成内容”的思路，也便于后续根据学习状态调整地图结构^{[10][12]}。

5.2.2 节点地图生成实现

节点地图生成在 Unity 本地完成。系统读取当前 MapConfigSO 后，会按列遍历 roomBlueprints，根据 min 和 max 随机生成本列节点数量，并把实例化后的节点保存到 columnGroups 中。随后，系统按照相邻列关系建立连线，并把连线保存到 allConnections 中。和复杂地形生成相比，节点式地图更接近 Roguelike 类关卡路径的

组织方式，它实现成本相对较低，同时也能提供多路线选择和重复体验^[14]。

生成房间类型时，首列优先设置为普通房间，末列设置为 Boss 房间，中间列则根据配置或概率生成普通房间、精英房间和休息房间。这样处理以后，玩家可以从压力较低的节点进入地图，每轮地图也都有一个明确的终点。

节点之间的连接关系使用邻接表保存。allConnections 的键是当前节点对象，值是该节点可以通向的下一组节点。地图渲染时，Unity 根据邻接关系绘制路线；玩家点击节点时，也通过这个关系判断节点是不是当前可到达范围内。该实现可以显示多分支路线，同时避免玩家直接跳到没有解锁的节点。

5.2.3 题目适配与降级处理实现

题目适配由房间类型和 difficulty 共同决定。普通房间主要匹配低难度题目，精英房间匹配中等难度题目，Boss 房间匹配高难度题目，休息房间不加载题目。Unity 进入战斗节点后，优先使用/api/game/questions 返回的在线题库，并按目标难度筛选题目。

当网络请求失败、后端返回空题库或游客模式无法取得在线数据时，Unity 使用本地题库作为降级来源。本地题库同样按照 difficulty 保存题目，使玩家即使在网络异常情况下也能继续完成基础流程。在线题库保证题目内容可以由后台维护，本地题库则保证原型系统具备基本可用性。

5.2.4 节点访问与进度保存实现

节点访问控制通过当前逻辑位置和可达节点集合实现。游戏开始时 currentPlayerRoom 为空，系统只允许玩家进入第一列可达节点；玩家完成某个节点后，currentPlayerRoom 会更新为该节点，后续可点击范围由 allConnections 中的下一组节点决定。

该机制可以保证玩家按照地图路径逐步推进，避免绕过前置任务直接进入后续节点。对于登录用户，游戏结果、学力值、等级、成就和题目作答明细，会通过/api/game/result、/api/user/study-ability、/api/achievements/sync、/api/questions/stats 和 /api/questions/attempts 同步到后端；对于游客用户，Unity 则使用 PlayerPrefs 保存本地累计数据。

通过上述实现，Unity 玩家端既能完成地图和战斗的即时表现，也能把关键学习

行为接入后端数据闭环。地图生成本身保持在本地执行，难度配置、题库内容和学习结果由后端提供和保存，从而减少客户端逻辑和业务数据之间的耦合。

5.3 Spring Boot 后端实现

Spring Boot 后端按业务职责划分为认证模块、游戏结果模块、学力值与等级模块、题库模块、成就模块、题目统计模块和管理后台模块。后端统一返回 JSON 响应，使用 success、message 和 data 表示请求结果、提示信息和业务数据。

表 5-3 后端玩家接口实现表

模块	接口	请求方式	实现功能
认证模块	/api/auth/login、/api/auth/register	POST	登录注册并返回 token 和用户状态
游戏会话	/api/game/session	POST	保存兼容旧版的游戏会话数据
游戏结果	/api/game/result	POST	保存本局游戏结果并计算学力值变化
难度配置	/api/game/difficulty	GET	返回学力值区间和 difficultyLevel
题库模块	/api/game/questions	GET	按 difficulty 返回启用题目
学力值模块	/api/user/study-ability	PUT	保存客户端同步的 studyAbility
等级模块	/api/user/level	PUT	保存等级经验并返回战斗数值
成就模块	/api/achievements、 /api/achievements/sync	GET/POST	读取和批量同步用户成就
题目统计	/api/questions/stats	POST	更新题目尝试次数和正确率
做题明细	/api/questions/attempts	POST	保存单题作答来源、结果和关联场次
错题复习	/api/review/sessions、 /api/review/questions	GET/POST	创建复习场次并获取复习题目

后端实现的关键在于保持数据处理职责集中。Unity 端只需要按照接口提交数据，后端负责校验 token、保存业务记录、更新用户状态和返回统计结果。管理后台接口位于 /api/admin/** 下，用于题库、难度配置、玩家、游戏记录和统计报表管理。

5.4 MySQL 数据库实现

MySQL 数据库保存系统运行中的基础数据和过程数据。数据库实现不只用于存储页面表单，还服务于学力值更新、题目统计、成就同步、错题复习和后台分析。结合系统真实数据库结构，核心数据表如表 5-4 所示。

表 5-4 MySQL 核心数据表实现

表名	主要字段	实现作用
users	id、username、password_hash、email、study_ability、current_level、experience、status	保存玩家账号、学力值和成长状态
admin_users	id、username、password_hash、role	保存管理员账号和角色
difficulty_config	id、difficulty_level、study_ability_min、study_ability_max、description	保存学力值到难度的映射规则
questions	id、question_text、correct_answer、wrong_answers_json、difficulty、enabled、deleted	保存题库内容和题目难度
game_results	user_id、play_time、rooms_completed、correct_answers、wrong_answers、difficulty、is_completed、gold_collected、created_at	保存游戏结算结果
game_sessions	user_id、play_time、progress、difficulty、session_date	保存兼容旧版的游戏会话与进度数据
review_sessions	user_id、difficulty、status、created_at、updated_at	保存错题复习场次和复习状态
user_question_attempts	user_id、question_id、is_correct、difficulty、source_type、game_result_id、review_session_id、answered_at	保存用户单题作答明细，并区分游戏和复习来源
user_achievements	user_id、achievement_id、current_value、is_unlocked、unlocked_at、updated_at	保存用户成就进度

数据库实体关系如图 5-2 所示。users 是玩家学习数据的中心表，它和 game_results、game_sessions、review_sessions、user_question_attempts、user_achievements 之间形成一对多关系，用来连接游戏结算、复习场次、做题明细和成就进度。

题目数据主要由 questions、question_stats 和 user_question_attempts 一起支撑。questions 保存题目内容和难度，question_stats 记录题目的总尝试次数和正确次数，user_question_attempts 记录某个用户单题作答过程，并通过 source_type 区分 GAME 和 REVIEW，再通过 game_result_id 或 review_session_id 关联普通游戏或错题复习来源。

数据库通过唯一约束、外键、CHECK 约束和索引来保证数据可用性。唯一约束用于避免账号重复，外键用于维持用户、题目、游戏结果和复习场次之间的关系，CHECK 约束用于限制 study_ability、difficulty、status 等字段的取值范围，索引用于支持后台分页查询、玩家时间线、题目质量分析和统计看板展示。

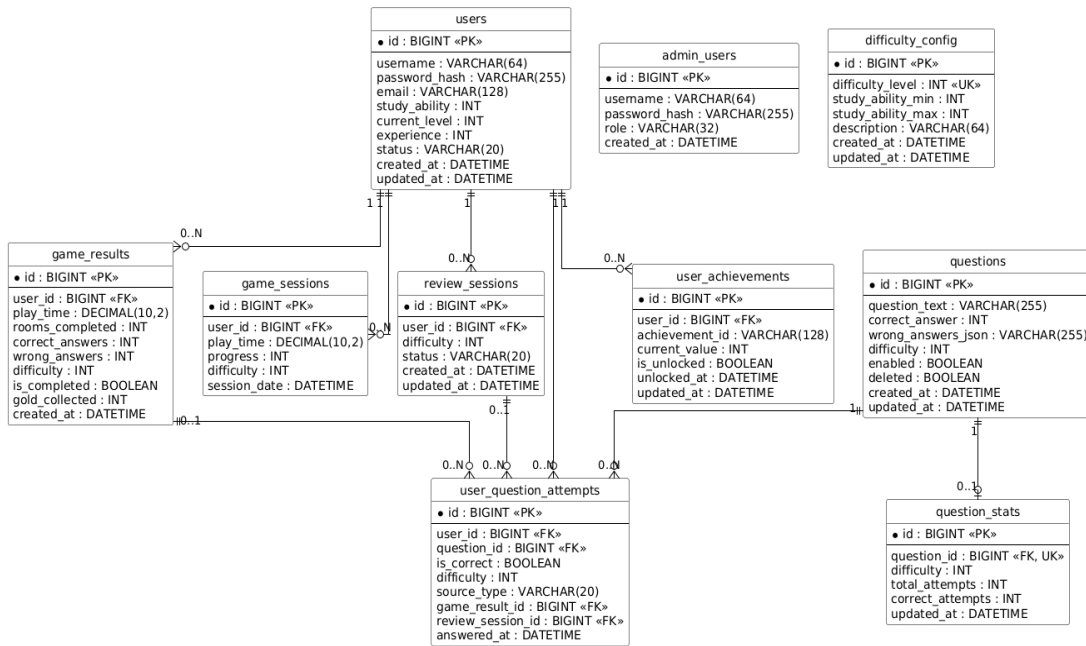


图 5-2 数据库 E-R 图

5.5 学力值更新算法实现

学力值更新算法是系统实现中的核心内容。根据项目中 EduGameService 的实现，游戏结束后，`/api/game/result` 接口会保存本局游戏结果，并调用 `updateStudyAbilityAfterGame` 方法更新用户的 `studyAbility`。该方法以答题正确率、房间完成度、金币完成度和时间效率作为输入指标，通过加权求和得到 `performanceScore`；随后根据答题数量计算 `confidence`，对学力值变化量进行缩放，并使用单局变化上限控制学力值波动。

在具体计算中，用户原学力值会作为更新基础，系统根据本局表现得到变化量，再把变化后的数值限制在合理范围内。

设用户原学力值为 A ，本局难度为 d ，正确题数为 C ，错误题数为 W ，完成房间数为 R ，收集金币数为 G ，游戏耗时为 P ，本局通关状态为 F ，当游戏通关时 $F = 1$ ，未通关时 $F = 0$ 。项目代码中的限幅函数可表示为：

$$\text{clamp}(x, l, u) = \min(\max(x, l), u) \quad (\text{公式 5.1})$$

答题总数与正确率分数计算如下：

$$N = C + W \quad (\text{公式 5.2})$$

$$S_a = \begin{cases} 0, N = 0 \\ \frac{C}{N}, N > 0 \end{cases} \quad (\text{公式 5.3})$$

房间、金币和时间指标的目标值与当前难度 d 相关。真实代码中的目标函数为：

$$T_r(d) = 8 + 2 \max(0, d - 1) \quad (\text{公式 5.4})$$

$$T_g(d) = 100 + 50 \max(0, d - 1) \quad (\text{公式 5.5})$$

$$T_t(d) = 240 + 60 \max(0, d - 1) \quad (\text{公式 5.6})$$

其中 $T_r(d)$ 表示目标房间数, $T_g(d)$ 表示目标金币数, $T_t(d)$ 表示目标耗时秒数。

三个辅助分数分别为：

$$S_r = \text{clamp}\left(\frac{R}{T_r(d)}, 0, 1\right) \quad (\text{公式 5.7})$$

$$S_g = \text{clamp}\left(\frac{G}{T_g(d)}, 0, 1\right) \quad (\text{公式 5.8})$$

$$S_t = \begin{cases} 0, P \leq 0 \\ \text{clamp}\left(\frac{T_t(d)}{P}, 0, 1\right), P > 0 \end{cases} \quad (\text{公式 5.9})$$

若本局未通关，则时间效率得分进一步修正为：

$$S_t' = \min(S_t, 0.30) \quad (\text{公式 5.10})$$

若本局通关，则 $S_t' = S_t$ 。

同时，未通关的惩罚计算为：

$$P_i = \begin{cases} 0, F = 1 \\ (1 - S_r) \times 0.20, F = 0 \end{cases} \quad (\text{公式 5.11})$$

综合表现分采用加权求和，权重与代码常量 ACCURACY_WEIGHT、ROOM_WEIGHT、GOLD_WEIGHT 和 TIME_WEIGHT 保持一致：

$$S_p = 0.55S_a + 0.20S_r + 0.15S_g + 0.10S_t' - P_i \quad (\text{公式 5.12})$$

为了避免少量题目导致学力值剧烈变化，系统使用答题数量计算置信度：

$$\gamma = \min\left(\frac{N}{10}, 1\right) \quad (\text{公式 5.13})$$

学力值变化量先由综合表现分换算为原始变化量，再经过置信度缩放和单局变化上限约束：

$$\Delta_{\text{raw}} = \text{round}((S_p - 0.5) \times 20) \quad (\text{公式 5.14})$$

$$\Delta = \begin{cases} \text{clamp}(\text{round}(\Delta_{\text{raw}} \times \gamma), -6, 8), F = 1 \\ \text{clamp}(\text{round}(\Delta_{\text{raw}} \times \gamma), -6, 1), F = 0 \end{cases} \quad (\text{公式 5.15})$$

最终学力值为：

$$A' = \text{clamp}(A + \Delta, 1, 100) \quad (\text{公式 5.16})$$

其中，-6 表示单局最大下降值。如果本局通关，单局最大提升值为 8；如果本局没有通关，单局最大提升值限制为 1。这样处理可以避免玩家未完成关卡时，因为某些局部指标较高而让学力值大幅上升，使失败结算更接近实际学习表现。

该算法把多个学习行为指标合并成一个比较容易解释的成长数值。正确率决定主要变化方向，房间完成、金币获得和耗时作为辅助因素；**confidence** 用于减小少量答题带来的偶然影响，**clamp** 操作用于控制学力值在单局中的变化范围。难度平滑逻辑进一步保证了学力值变化不会马上造成过大的难度跳跃，使系统自适应调整过程相对稳定。这个设计和动态难度调整、心流体验相关研究中“挑战强度应随用户状态变化并保持平衡”的观点基本一致^{[6][7][9]}。

5.6 题目战斗与统计实现

题目战斗的数据入口是 `/api/game/questions?difficulty={difficulty}`。后端从 `questions` 表中筛选指定 `difficulty` 且启用的题目，返回 `questionId`、`questionText`、`correctAnswer`、`wrongAnswers` 和 `difficulty`。Unity 使用 `QuestionData` 接收题目数据，并在战斗界面中展示题干和选项。题目战斗与统计交互如图 5-3 所示。

题目统计由 `/questions/stats` 完成。玩家完成单题后，Unity 上报 `questionId`、`isCorrect` 和 `difficulty`；后端根据 `questionId` 查找或创建 `question_stats` 记录，更新 `totalAttempts` 和 `correctAttempts`，并计算 `correctRate` 后返回。该模块使系统不仅能在 Unity 中给出战斗反馈，还能在后端沉淀题目表现数据。

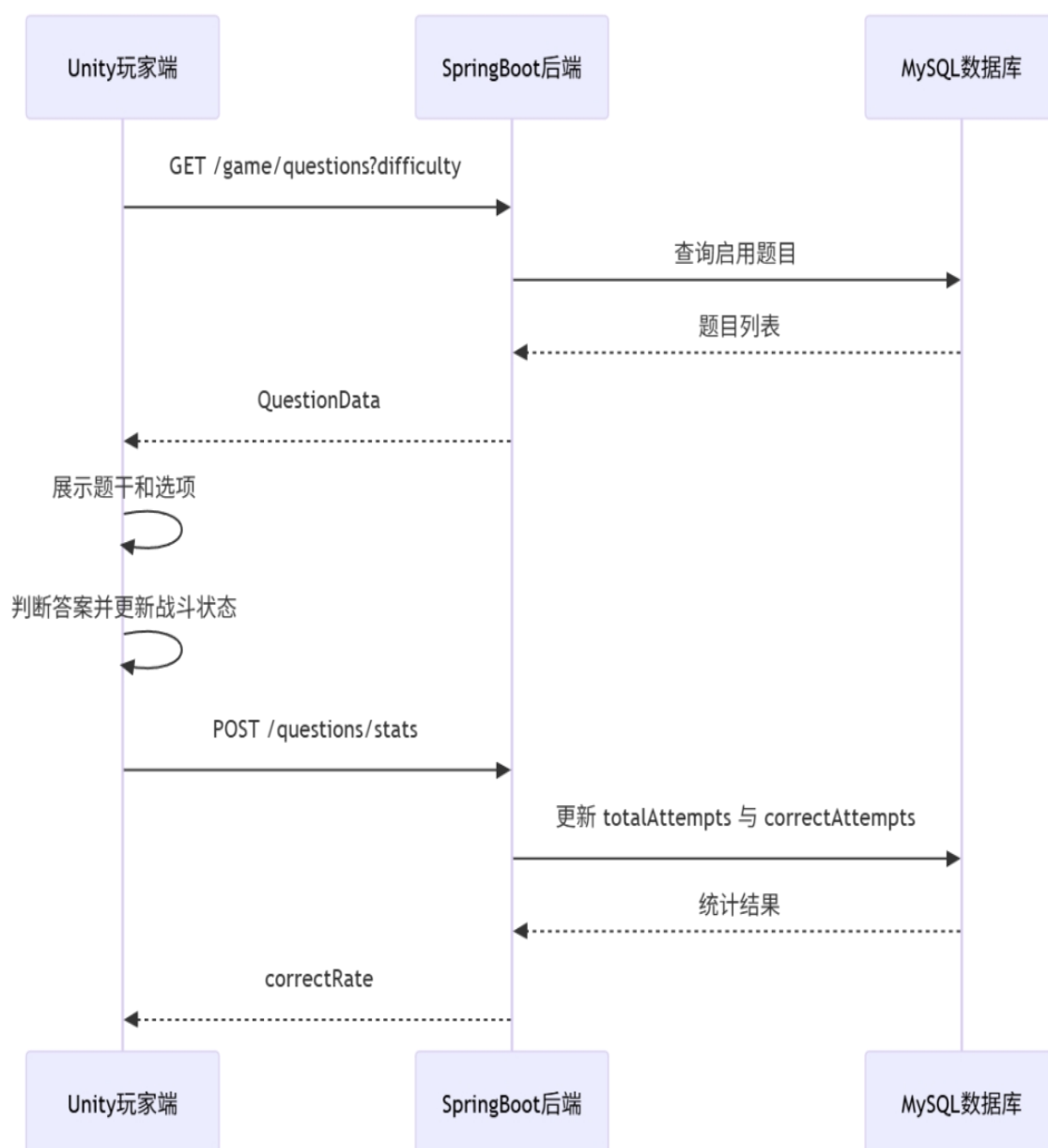


图 5-3 题目战斗与统计时序图

5.7 成就与游戏结果回传实现

成就模块由 GET /achievements 和 POST /achievements/sync 组成。登录后，Unity 获取当前用户的成就进度；游戏结束或达成条件后，Unity 批量同步 achievementId、currentValue 和 isUnlocked。后端同步时采用幂等合并策略：若同一用户同一成就已存在，则 currentValue 保留较大值；isUnlocked 一旦为 true 不再被 false 覆盖；unlockedAt 记录首次解锁时间。

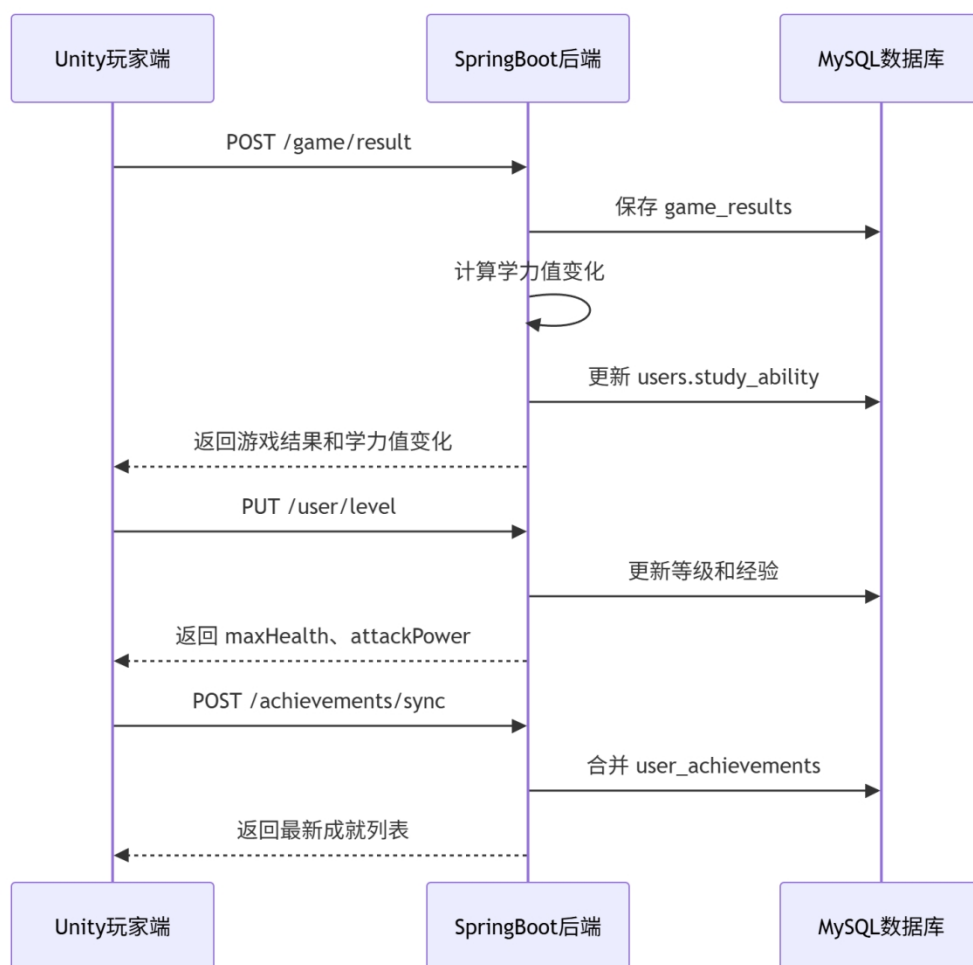


图 5-4 游戏结果与成就同步时序图

通过成就同步和游戏结果回传，系统把短期战斗表现、长期成长反馈和后台统计连接起来，形成可持续更新的数据链路。

5.8 Vue 管理后台实现

Vue 管理后台使用 Vue 3、Vue Router、Pinia、Element Plus 和 ECharts 实现，主要给管理员使用。它在系统中主要承担配置维护和数据查看的作用。后台通过 /api/admin/** 接口访问后端，可以完成题库维护、难度配置、玩家管理、游戏记录查询和统计分析等操作。

管理后台的作用，主要体现在让系统闭环更容易维护。管理员可以通过题库管理调整学习内容，通过难度配置调整 studyAbility 和 difficulty 之间的对应关系，也可以通过统计分析查看题目和玩家表现。这些数据后续可以作为优化地图交互和学习策略的参考。

5.9 本章小结

本章围绕教育游戏地图交互引擎的具体实现进行说明,重点介绍了 Unity 玩家端、Spring Boot 后端、MySQL 数据库和 Vue 管理后台之间的配合方式。Unity 玩家端负责登录认证、难度获取、节点地图生成、题目战斗、奖励反馈和数据同步,让学习任务可以用游戏关卡形式展示出来。Spring Boot 后端负责用户认证、题库查询、游戏结果保存、学力值更新、成就同步和题目统计,使学习行为可以被记录、计算和再次使用。MySQL 数据库保存用户、题目、游戏结果、做题明细、成就和复习数据,为数据闭环提供持久化支持。Vue 管理后台提供题库、难度、玩家和统计管理功能,使系统内容和规则能够后续维护。

通过上述实现,本文将第四章提出的“学习行为采集—学力值更新—难度配置匹配—节点地图生成—题目战斗—结果回传”设计流程落到了具体代码和数据结构中。学力值更新算法把正确率、完成房间数、金币收集和耗时等指标转化为用户能力变化;节点地图生成逻辑根据难度配置调整地图结构;题目战斗模块把答题结果转化为攻击、扣血和结算反馈;成就与统计模块则把短期游戏结果沉淀为长期学习记录。

第 6 章 系统测试

6.1 测试环境

为保证测试结果能够反映系统在实际开发与运行环境下的表现，本文在本地开发环境中完成系统部署与功能测试。测试环境包括 Unity 客户端、Spring Boot 后端服务、Vue 管理端和 MySQL 数据库。具体测试环境如表 6-1 所示。

表 6-1 测试环境表

测试项目	环境说明
操作系统	Windows 11
客户端开发环境	Unity 2022.3 LTS, Visual Studio, VS Code
后端开发环境	JDK 8, Spring Boot, Maven
管理端开发环境	Vue 3, Node.js, Element Plus
数据库环境	MySQL 8.0
浏览器环境	Microsoft Edge, Google Chrome
接口测试工具	浏览器开发者工具、Postman 和系统页面实际操作
测试方式	功能测试、流程测试、界面运行结果验证

测试过程中，学生端主要通过 Unity 客户端进行功能验证，管理端主要通过浏览器访问后台管理页面进行验证。后端服务负责接收学生端和管理端请求，并与数据库进行数据交互。测试重点在于验证系统主要功能是否可用、数据是否能够正确保存和查询，以及学生端学习流程是否能够形成完整闭环。

6.2 测试用例设计

系统测试采用黑盒测试为主，测试人员从用户操作角度出发，验证输入、操作流程和输出结果是否符合预期。根据系统需求分析，测试用例分为学生端功能测试、管理端功能测试和核心业务流程测试三类。

从表 6-2 可以看出，测试用例覆盖了系统的主要功能入口和核心业务链路。学生端测试重点验证学习交互流程是否完整；管理端测试重点验证后台数据维护和统计查看功能是否可用；核心流程测试重点验证答题行为、结果保存、学力值更新和错题复习之间的数据衔接是否正常。

表 6-2 系统功能测试用例

测试编号	测试模块	测试内容	测试步骤	预期结果	实际结果	结论
TC-01	学生端	用户注册	输入学生账号、密码等信息并提交注册	系统提示注册成功，用户信息保存到数据库	与预期一致	通过
TC-02	学生端	用户登录	输入正确账号和密码并点击登录	登录成功，进入学生端主界面	与预期一致	通过
TC-03	学生端	节点地图生成	登录后进入节点地图页面	系统根据难度配置生成节点地图，并显示可挑战节点	与预期一致	通过
TC-04	学生端	关卡答题	选择地图节点进入关卡并提交答案	系统判断答案正确性，并触发对应战斗反馈	与预期一致	通过
TC-05	学生端	游戏结果保存	完成关卡答题并进入结算流程	系统保存得分、胜负状态、答题情况等结果数据	与预期一致	通过
TC-06	学生端	错题复习	进入错题复习页面并开始复习	系统获取历史错题并创建复习会话	与预期一致	通过
TC-07	管理端	管理员登录	输入管理员账号和密码并点击登录	登录成功，进入后台管理首页	与预期一致	通过
TC-08	管理端	题库管理	新增、修改、删除或查询题目信息	题目信息能够正确保存、更新或展示	与预期一致	通过
TC-09	管理端	难度配置管理	修改学力值区间和对应难度等级	难度配置保存成功，并可被学生端读取	与预期一致	通过
TC-10	管理端	玩家数据管理	查询学生用户数据和学习状态	系统展示玩家基础信息、学力值和相关学习数据	与预期一致	通过
TC-11	管理端	游戏结果管理	查询学生关卡结果记录	系统展示关卡得分、胜负状态和完成时间等信息	与预期一致	通过
TC-12	管理端	统计看板	进入统计看板页面查看统计数据	系统以表格或图表形式展示题目统计和学习结果	与预期一致	通过
TC-13	核心流程	学力值更新	学生完成一次关卡答题并结算	系统根据答题表现更新学生学力值	与预期一致	通过
TC-14	核心流程	失败结算学力值更新	学生未通关并上传失败结算数据	系统限制时间得分，加入未完成惩罚，并按失败口径更新学力值	与预期一致	通过
TC-15	核心流程	错题记录生成	学生在关卡中答错题后进入错题复习	答错题进入复习题目列表	与预期一致	通过
TC-16	核心流程	题目统计上报	学生完成题目作答后查看管理端统计数据	管理端能够展示题目作答次数、正确情况等统计结果	与预期一致	通过

6.3 系统运行结果展示

6.3.1 学生端运行结果展示

学生端是系统的主要交互入口，承担学生登录注册、节点地图探索、关卡答题、结果结算和错题复习等功能。测试过程中，学生用户首先完成登录或注册操作，随后进入节点地图页面。系统根据学生当前学习状态和难度配置生成节点式地图，学生可选择已解锁节点进入关卡答题。



图 6-1 玩家登录与注册界面

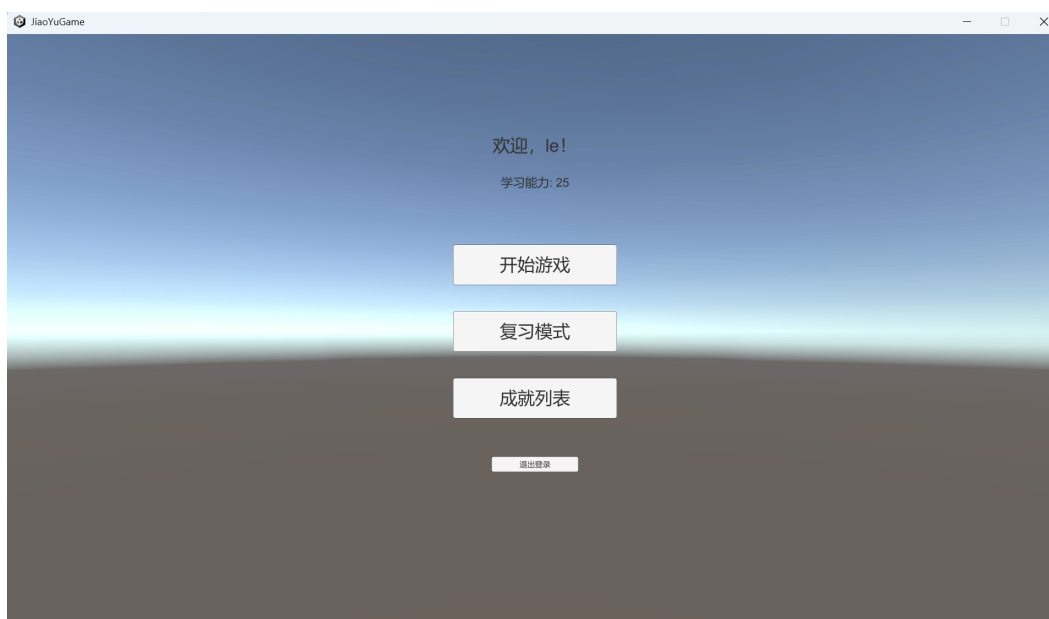


图 6-2 玩家端用户主页

如图 6-1 所示，学生用户可通过登录注册页面进入系统。结合测试用例 TC-01 和 TC-02 可知，系统能够完成学生账号创建和身份验证，登录成功后能够进入学生端主界面，为后续地图生成和关卡答题流程提供身份基础。

如图 6-2 所示，学生用户登录成功后进入玩家端用户主页。主页作为学生端主要功能入口，能够集中展示地图挑战、关卡学习、错题复习和个人学习状态等功能入口。结合测试用例 TC-02 可知，系统能够在身份验证通过后正确跳转到玩家端主页，为后续学习地图和答题流程提供导航入口。



图 6-3 节点式地图生成界面

如图 6-3 所示，系统能够生成节点式学习地图，并通过节点和连线展示学习路径。结合测试用例 TC-03 可知，节点地图生成功能能够正常运行，学生可以根据地图中展示的节点状态选择可挑战关卡。使低学力玩家获得较清晰的路径，高学力玩家获得更多路线分支和挑战节点。该界面体现了学习行为数据对地图结构的影响。

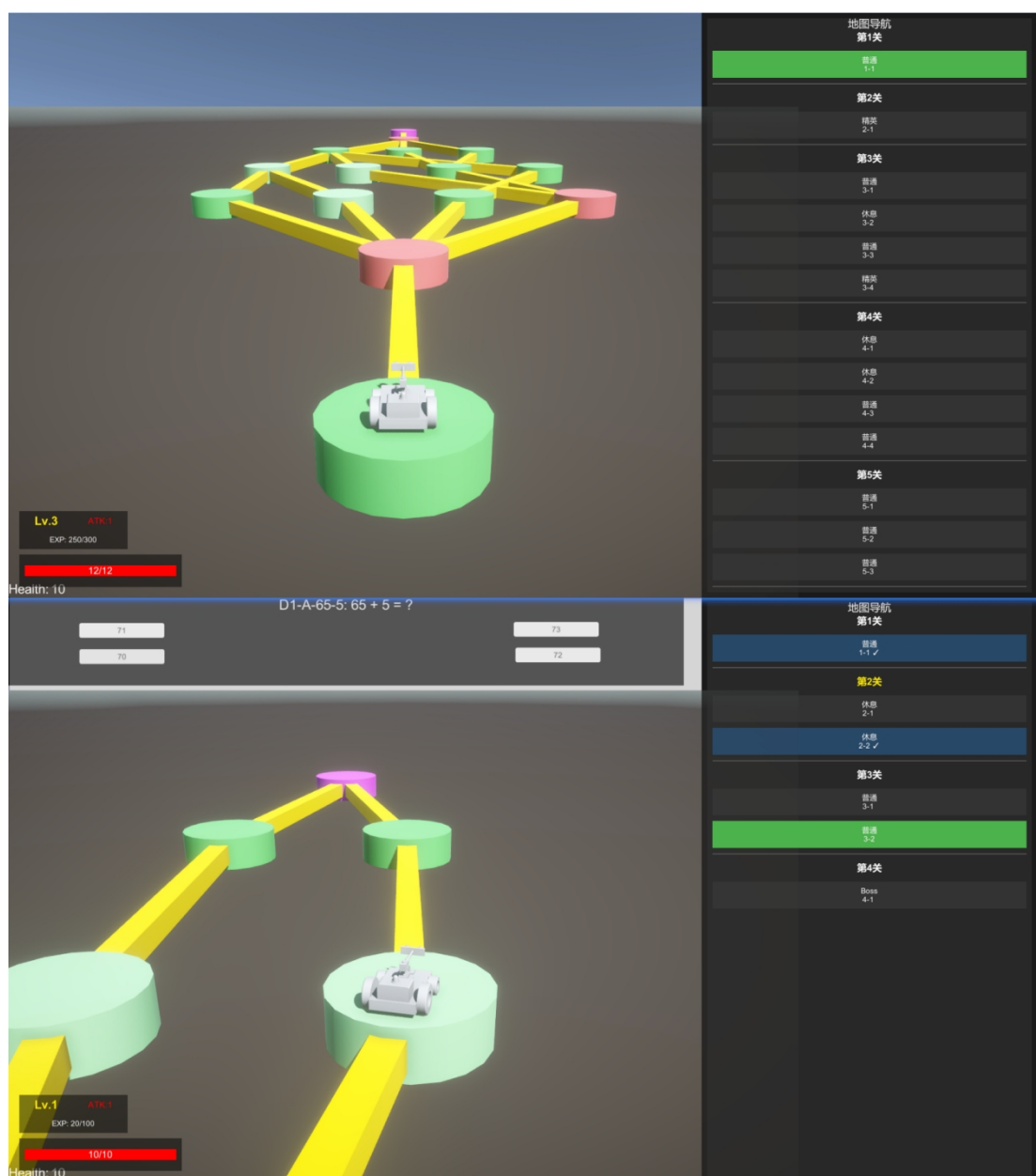


图 6-4 题目战斗界面

如图 6-4 所示，学生进入关卡后，系统将题目作答与战斗反馈结合。学生提交答案后，系统判断答案正确性，并根据答题结果触发攻击或受击反馈。结合测试用例 TC-04 可知，关卡答题功能能够正常执行，学习行为能够直接影响游戏过程。

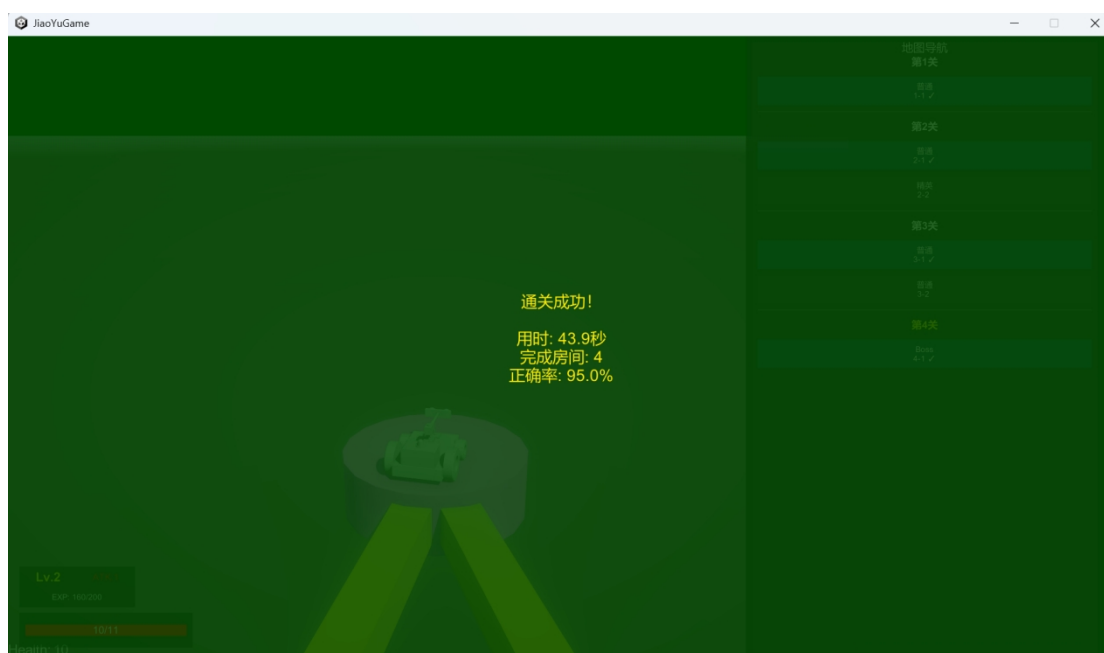


图 6-5 游戏结算与奖励成就界面

如图 6-5 所示，关卡结束后系统能够展示本次游戏结果，包括得分、胜负状态和奖励反馈等信息。结合测试用例 TC-05 和 TC-13 可知，系统能够保存本次游戏结果，并根据答题表现更新学生学力值，实现答题结果与学习状态之间的关联。

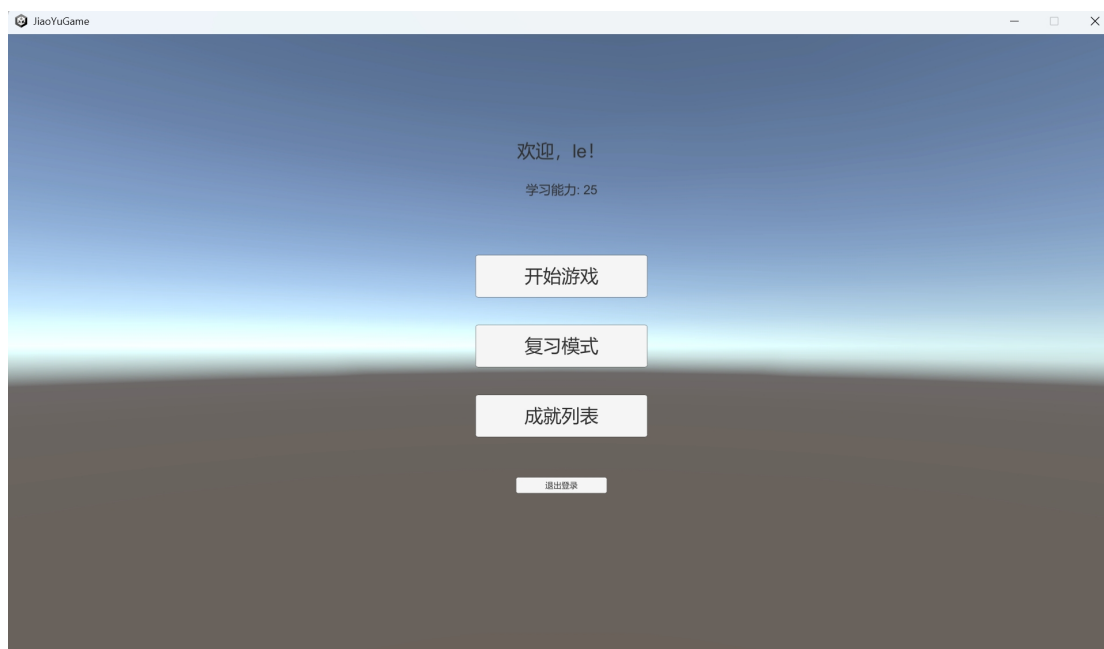


图 6-6 玩家端比赛后学力值变化

比赛后学力值变化测试如图 6-6 所示。测试用户进入游戏前主页显示学力值为 20，完成一局游戏并上传结算数据后，再次返回主页时学力值更新为 25。该结果说明客

户端能够读取后端更新后的 studyAbility 字段，并将比赛后的学力值变化同步显示在用户主页中。

6.3.2 管理端运行结果展示

管理端主要面向管理员，用于完成题库维护、难度配置、玩家数据查看、游戏结果管理和统计看板展示等功能。管理端不直接参与学生答题过程，但为学生端运行提供题目数据、难度配置和结果统计支持。

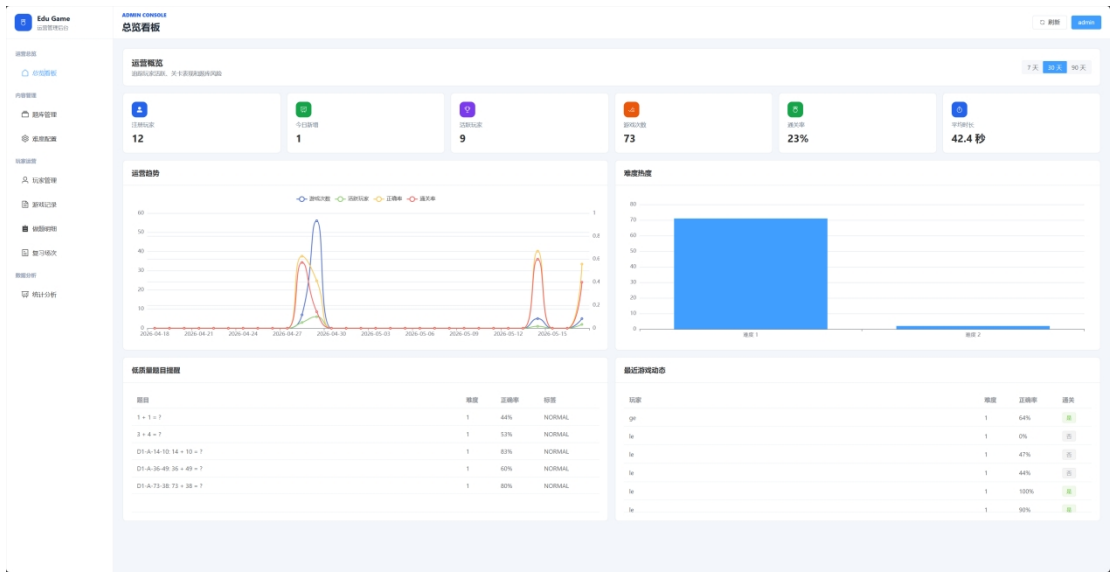


图 6-7 管理后台总览看板

如图 6-7 所示，管理后台总览看板用于展示系统运行过程中的统计信息。管理员可以查看注册玩家、活跃玩家、游戏次数、平均游戏时长、通关率、题目正确率和成就解锁情况等数据。该界面说明后端不仅为玩家端提供实时接口，也能够将游戏结果、题目统计和成就数据整理为管理端可观察的信息。

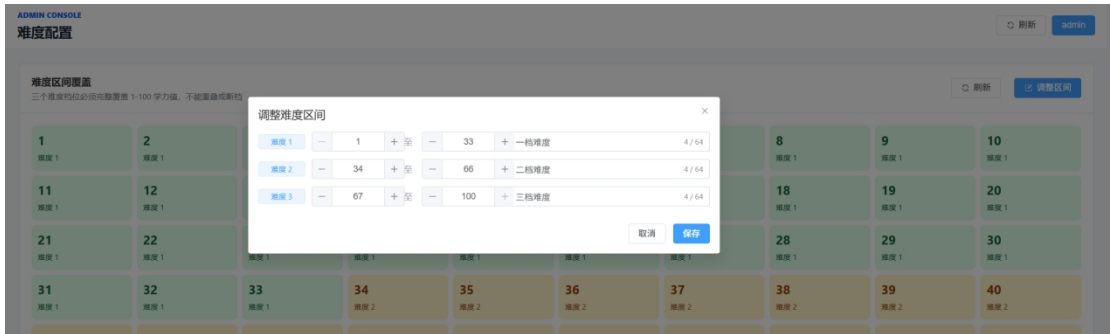


图 6-8 难度配置界面

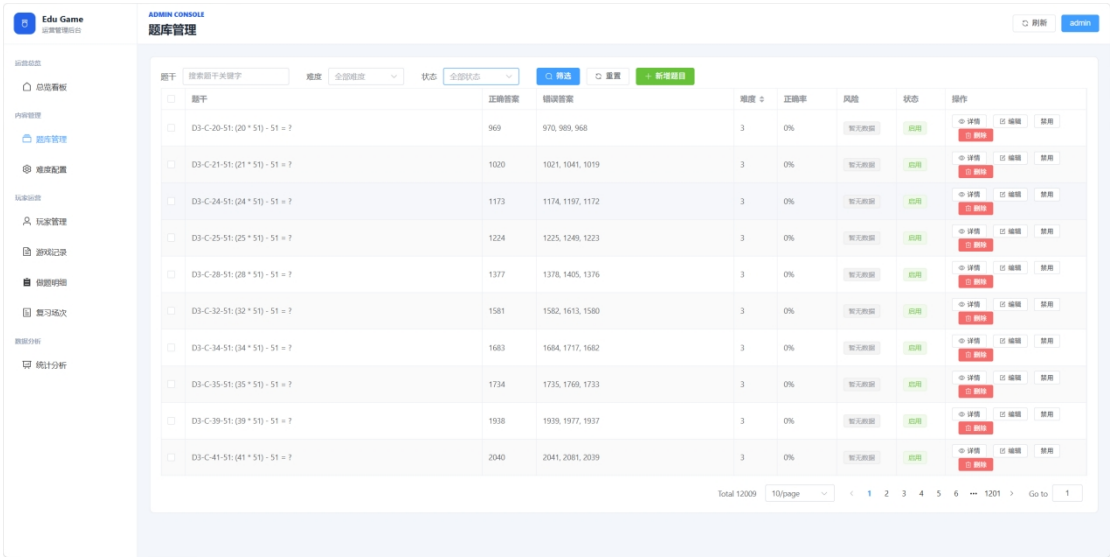


图 6-9 题库管理界面

如图 6-8 和图 6-9 所示，题库管理与难度配置界面用于支撑系统内容维护。管理员可以维护题干、答案、题目难度和启用状态，也可以调整学力值区间与 difficultyLevel 的对应关系。由于难度配置由后端 difficulty_config 表维护，系统后续调整难度策略时不需要重新修改 Unity 客户端，从而提高系统可维护性。

6.4 测试结果分析

从功能测试结果来看，学生端的登录注册、节点地图生成、关卡答题、游戏结果保存和错题复习等功能，基本都能按预期运行。学生登录后可以进入节点地图页面，系统会根据难度配置生成地图节点；学生进入关卡后，可以完成题目作答，并根据答题结果得到战斗反馈；关卡结束以后，系统能够保存游戏结果，并更新学生学力值；当学生产生错题记录后，错题复习模块也能够生成对应的复习内容。

从管理端测试结果来看，管理员登录、题库管理、难度配置管理、玩家数据管理、游戏结果管理和统计看板等功能可以正常使用。后台管理端能够给学生端提供题目数据和难度配置支持，也能够查看学生端产生的游戏结果和题目统计数据。这说明系统前后端之间的数据交互基本正常，管理端可以起到内容维护和数据查看作用。

从核心业务流程来看，系统能够形成“学生答题—结果保存—学力值更新—地图与难度变化—错题复习”的基本闭环。学生答题行为不只用于关卡结算，还会继续影响学力值和后续学习路径。错题记录能够进入复习流程，题目作答数据也能在管理端

统计展示。这说明系统基本实现了学习行为数据和教育游戏反馈机制之间的联动。

不过，本次测试主要集中在功能可用性和核心流程正确性方面，还没有进行大规模并发测试，也没有做长期学习效果跟踪实验。因此，系统目前可以验证原型功能和业务流程的完整性，但对于大量用户同时访问、长期学习表现变化以及学习效果提升程度，还需要在后续使用中继续测试和分析。

6.5 本章小结

本章对教育游戏地图交互引擎系统进行了测试。首先说明了系统测试环境，包括 Unity 客户端、Spring Boot 后端、Vue 管理端和 MySQL 数据库等运行条件；然后设计了覆盖学生端、管理端和核心业务流程的功能测试用例；接着结合系统运行截图，展示了登录注册、节点地图生成、关卡答题、结果结算、错题复习、题库管理、难度配置和统计看板等功能的运行情况；最后对测试结果进行了分析。

测试结果说明，系统主要功能可以正常运行，学生端学习交互流程、管理端数据维护流程，以及后端核心数据流转过程，基本能够满足需求。系统可以根据学生答题行为保存学习结果、更新学力值、生成错题复习内容，并在管理端展示相关统计数据，基本达到了教育游戏地图交互和学习反馈设计的预期目标。

第7章 结论

本文围绕教育游戏软件中的三维地图交互与学习行为反馈问题，设计并实现了一个学习行为驱动的地图交互引擎原型系统。系统以节点式拓扑地图作为关卡组织方式，在 Unity 三维场景中呈现学习路径、关卡节点和奖励反馈，并通过 Spring Boot 后端记录学生答题结果、关卡完成情况、耗时、奖励和错题等学习行为数据。后端根据这些数据计算用户学力值，并将学力值映射为不同难度档位，使地图路线数量、Boss 节点数量、奖励节点分布和题目难度能够随着学生学习状态发生变化，从而形成“学习行为采集—学力值更新—难度映射—地图生成—题目战斗—结果回传”的闭环流程。

在系统实现方面，本文基于 Unity、Spring Boot、MySQL 和 Vue 管理后台构建了较完整的原型系统。Unity 玩家端主要完成登录注册、节点地图生成、关卡进入、题目战斗、奖励反馈、成就展示和数据同步；Spring Boot 后端主要完成用户认证、题库查询、游戏结果保存、学力值更新、成就同步、题目统计和错题复习等功能；MySQL 数据库负责保存用户、题目、难度配置、游戏结果、作答记录、成就进度和复习数据；Vue 管理后台则提供题库维护、难度配置、玩家数据管理和统计查看等功能。通过功能测试和运行结果展示可以看出，系统主要业务流程能够正常运行，学生端答题行为能够被记录并影响后续学力值和地图难度，管理端也能够查看和维护相关数据，基本验证了本文设计方案的可行性。

从毕业设计角度看，本文不仅完成了教育游戏地图交互引擎的原型实现，也对学习行为数据如何参与游戏进程进行了方法层面的整理。本文将答题正确率、关卡完成情况、耗时、奖励和错题等行为数据转化为学力值，再通过学力值控制地图结构和题目难度，使学习行为不再只作为后台统计结果，而是成为地图生成、关卡挑战和奖励反馈的重要依据。该设计在一定程度上增强了教育游戏中学习任务与游戏反馈之间的联系，为后续教育游戏中的个性化学习路径设计和动态难度调整提供了一种可参考的实现思路。

当然，当前系统仍然存在一定不足，后续还可以从多个方面继续完善。首先，学力值计算模型目前主要依据答题正确率、关卡完成度、耗时和奖励等指标进行规则化计算，对学生长期学习状态、知识点掌握差异和题目难度区分的刻画还不够细致，后

续可以引入知识点维度、历史学习趋势和题目区分度等因素，提高学力值对学生真实能力变化的反映能力。其次，节点式地图生成仍主要依赖预设配置和随机规则，地图结构的多样性、路径平衡性和学习目标约束仍有优化空间，后续可以进一步结合知识点覆盖率、节点权重和关卡节奏控制来改进地图生成效果。再次，当前三维场景表现和管理端数据分析功能仍偏基础，后续可以完善地图镜头控制、节点动态高亮、奖励动画、知识点掌握分析、学习趋势统计和错题分布分析等功能。最后，本文测试主要集中在功能可用性和核心流程闭环方面，尚未开展大规模用户测试、并发性能测试和长期学习效果跟踪，后续可以在更真实的教学场景中进行应用验证，以进一步检验系统的实际使用价值。

总体来看，本文完成了一个以学习行为数据驱动地图交互和难度调整的教育游戏原型系统。系统将节点式地图生成、题目战斗、学力值更新、成就奖励和数据回传结合起来，使学习过程能够通过游戏化方式得到呈现和反馈。虽然系统在算法精细度、场景表现、数据分析和实证验证方面仍有改进空间，但已经基本实现了课题预期目标，并为后续教育游戏中自适应地图生成和个性化学习反馈研究提供了实践基础。

致 谢

时光荏苒，本科阶段的学习生活即将结束。回顾论文写作与课题研究的全过程，我深切感受到老师、同学和家人给予我的帮助与支持。在此，谨向所有关心和帮助过我的人表示诚挚的感谢。

首先，衷心感谢我的指导老师。在论文选题、结构安排、资料查阅、内容修改以及最终定稿过程中，奚雪峰老师始终给予我耐心细致的指导，帮助我理清研究思路，完善论文内容，使我在专业知识和论文写作能力方面都有了进一步提升。

同时，感谢学院各位任课老师在本科阶段对我的培养。正是老师们在专业课程中的悉心讲授，为我完成本次毕业论文奠定了坚实基础。感谢同学和朋友们在论文撰写过程中给予的交流、鼓励与帮助，使我能够更加顺利地解决遇到的问题。

最后，感谢我的家人一直以来的理解、支持与陪伴。他们的鼓励是我不断前进的重要动力。今后，我将带着这份感恩继续学习与成长，以更加踏实的态度面对未来的挑战。

参 考 文 献

- [1] 顾怡雯,张佳怡,张孔燕,等. 近五年国际游戏化学习研究综述[J]. 开放学习研究,2020,25(4):10-18.
- [2] 白书瑞,丘琪鸿. 探究游戏化课堂中学生的学习动机产出:系统文献综述[C]//第四届游戏化学习学术会议暨中国教育技术协会教育游戏专业委员会 2019 年会论文集. 2019:146-161.
- [3] 李佳平,施高俊,杨俊锋. 2016~2021 年基于游戏的数学学习实证研究系统综述[C]//中国高等教育学会学习科学研究分会 2021 学术年会论文集. 2021:348-354.
- [4] 郑春萍,徐畅,张娴,等. 近十年数字游戏应用于语言教学的系统性综述[J]. 现代教育技术,2021,31(6):41-48.
- [5] 孙竹墨,姜婷婷,陈佩龙,等. 为教育游戏用户创造心流体验:提升学习表现和满意度[J]. 现代情报,2026,46(2):4-17.
- [6] 李通. 数字化游戏的动态难度调整模型建构[D]. 山东:山东师范大学,2014.
- [7] 王广新,刘聪. 学生对动态难度游戏的享乐体验与成就目标取向选择[J]. 现代教育技术,2019(6):69-75.
- [8] 新田耕豊. 教育游戏中的难度自适应机制设计探究--基于《成语小秀才》的通关大数据[C]//第 24 届全球华人计算机教育应用大会(GCCCE 2020)论文集. 2020:111-113.
- [9] Diana Sofia Lora-Ariza, Antonio A. Sanchez-Ruiz, Pedro Antonio Gonzalez-Calero, et al. Measuring Control to Dynamically Induce Flow in Tetris[J]. IEEE transactions on GAMES,2022,14(4):579-588.
- [10] MARK HENDRIKX, SEBASTIAAN MEIJER, JOERI VAN DER VELDEN, et al. Procedural Content Generation for Games: A Survey[J]. ACM transactions on multimedia computing communications and applications,2013,9(1):1.1-1.22.
- [11] Summerville, Adam, Snodgrass, Sam, Guzdial, Matthew, et al. Procedural Content Generation via Machine Learning (PCGML)[J]. IEEE transactions on GAMES,2018,10(3):257-270.
- [12] MALEKI M F, ZHAO R. Procedural content generation in games: a survey with insights on emerging LLM integration[C]//Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2024,20(1):167-178.
- [13] 靳仕恒,郑春辉,吕德生. 面向游戏场景生成的细分插槽WFC算法研究[J]. 计算机工程与应用,2023,59(1):244-251.
- [14] 郑爱宇,蔡江辉,杨海峰. 基于流场寻路算法的RougeLike-DBG游戏的设计与实现[J]. 信息技术,2021(10):48-53.
- [15] 单诚,邱相彬,李芮. 教育游戏研究的进展综述[J]. 电脑知识与技术,2023,19(5):140-143.
- [16] 贺松年. 游戏关卡难度动态调整机制设计[J]. 数字技术与应用,2026,44(2):198-200.
- [17] 王鹏云. 基于PCG的关卡生成游戏系统设计与实现[D]. 哈尔滨:哈尔滨工业大学,2025.
- [18] 朱玉林,莫志荣,凌一洲. 指向初中化学概念学习的教育游戏开发——化学飞车[J]. 化学教育

（中英文）,2025,46(19):87-91.

- [19] Rose M, Antonaci A, Bonnet M, et al. Learning Success, Engagement and Fun: Evaluating an Educational Game in Higher Education as Replacement for a Homework Assignment[C]//Entertainment Computing - ICEC 2025. Cham: Springer, 2025:320-333.
- [20] Wattanapornprom W, Wipachainun C, Lertpinitamonkul T, et al. Procedural Content Generation for 2.5D Rogue-Lite Games: An Evolutionary Algorithm Approach[C]//2024 28th International Computer Science and Engineering Conference (ICSEC). IEEE, 2024:1-6.

附录 A 译文

通过测量控制感在俄罗斯方块中动态诱导心流

Diana Sofía Lora-Ariza, Antonio A. Sánchez-Ruiz, Pedro Antonio González-Calero, and Irene Camps-Ortueta

摘 要

动态难度调整（Dynamic Difficulty Adjustment, DDA）是一套旨在根据玩家表现自动调整电子游戏难度的技术。本文提出了一种结合心流理论（Flow Theory）与基于案例推理（Case-Based Reasoning, CBR）的 DDA 方法。本质上，我们致力于生成与先前在同等技能水平玩家中产生心流的游戏会话具有相似难度演变过程的游戏会话。我们提出了一种 CBR 方法，用于动态评估玩家的技能水平，并根据最近游戏状态的相对复杂性来调整游戏难度。

我们利用该方法为《俄罗斯方块》（Tetris）开发了一个 DDA 系统，并通过一项包含 40 名参与者的实验表明，经由验证的问卷调查证实，该 DDA 版本对感知到的心流体验具有可测量的显著影响。

关键词 动态难度调整（DDA），心流，人工智能，电子游戏

1 绪论

1.1 引言

近年来，电子游戏产业迅猛发展。2020 年，该产业创造了超过 1700 亿美元的收入，并预计在未来几年继续增长。这一增长激励了投资者为新游戏的开发提供额外资源，同时也促使游戏工作室探索新途径，以克服制作和维护高质量电子游戏所固有的困难。一款普通的 AAA 级电子游戏制作周期通常在 1 至 3 年之间，这不仅是因为创建所有资产和游戏机制需要时间，更是因为必须持续测试这些创意，以验证其有效性及游戏的娱乐性。因此，游戏设计师在电子游戏制作中扮演着基础性的角色。他们通常负责定义游戏关卡、谜题、控制方式、游戏机制、角色对话……尤其是游戏难度等级，这通常涉及为 AI 角色定义不同的行为集合，以便根据玩家的技能水平进行调用[1]。换言之，当今大多数电子游戏中可见的“智能”是试图预测不同玩家行为的结果，进而在制作阶段实施一套标准动作，以在每个预期场景中做出适当响应。目前，电子游戏中的大多数内容都是预定义的，是在开发过程中创建的，寄希望于其适合大多数玩家——既具挑战性又不过于困难，但这并非总能实现。事实上，玩家的反馈是设计师区分极度困难的任务与巨大挑战的重要工具[2]。

一款“智能”电子游戏应能够在设计师无法预料的场景中决定如何行动，并提供适当的响应[3]。特别是，大多数电子游戏中存在的经典预定义难度等级并不适合所有玩家，且游戏难度的演变可能与玩家技能的提升速度不一致，导致游戏变得过易或过难。从这个意义上说，难度不应被理解为一种静态属性，而是一个源于玩家与所提出的挑战之间互动的主观因素[4]。动态难度调整（DDA）是一套旨在根据玩家表现自动适应游戏难度的技术[5]。也就是说，系统持续评估玩家与游戏的互动，根据玩家当前的需求改变游戏难度，以保持玩家的参与度。使用 DDA 的一个显著优势是降低电子游戏制作成本，因为如果游戏能够自我调整，设计师和开发人员在试图预测所有可能情况时所需的精力就会减少[6]。因此，DDA 引起了电子游戏行业领先公司的关注。Lopes 和 Bidarra[7]调查了学术界和工业界在游戏和模拟中适应性技术的最新进展，并得出结论：适应性正在确立为一个快速成熟的领域，目前的进展在适应最佳挑战水平以及诸如乐趣、挫败感、可预测性、焦虑或无聊等情绪状态方面显示出良好的

结果。

心流（Flow）是一种精神状态，处于该状态的人在进行某项活动时会完全沉浸其中，感到专注、全情投入和享受，通常会导致时间感知的改变。心流与电子游戏沉浸感之间的联系非常直接，因为将玩家带入其个人心流区域的能力已被认定为设计精良的游戏的一项要求[8]。

本文提出了一种新颖的 DDA 方法，该方法利用了心流理论[9]和基于案例推理（CBR）[10]的思想。使用心流理论的一个关键优势在于，由于它是心理学研究中非常流行的理论，因此存在不同的工具来测量它，包括基于生理测量和事后问卷调查的方法。这些工具可用于识别先前在玩家中产生心流的游戏。我们的方法旨在通过改变游戏难度来生成心流，使其演变过程类似于先前在具有相似技能水平的玩家中产生心流的游戏。我们使用 CBR 方法来动态评估玩家的技能水平，并利用基于当前游戏及一组先前相似标记游戏的相对复杂性的控制度量来调整游戏难度。我们为《俄罗斯方块》实现了一个 DDA 系统，并使用经过验证的心流问卷表明，该系统对玩家产生了可测量的影响。

本文后续内容结构如下：下一章节将对心流理论进行概述，并说明其与游戏动态难度调整之间的关联。第三章节则以《俄罗斯方块》为实例，阐述基于案例推理的动态难度调整方法。第四章节介绍了一项实验研究，结果证实该方法能够有效增强玩家的心流体验。文章最后对研究结论及未来工作方向进行了总结。

2 相关工作

2.1 心流理论

Csikszentmihalyi[9]把心流界定为一种极致的体验状态，在这种状态下，其他一切事物都变得无关紧要。能否进入心流状态，关键在于个体所感知到的任务挑战与自身能力之间能否达成恰到好处的平衡。Nakamura 等学者[11]提出，决定个体体验质量的并非客观层面的挑战难度与技能水平，而是主观认知中的挑战与技能。换句话说，个体唯有对成功完成任务抱有信心，才有可能步入心流状态。Csikszentmihalyi 归纳出九个维度，这些维度共同刻画了心流的理想心理状态[12]：挑战与技能的均衡、

行动与意识的统一、目标清晰明确、反馈及时准确、全神贯注于当前任务、掌控感、自我意识的消解、时间感的扭曲，以及自成目的的体验。

Nakamura 等人[11]指出，注意力在实现持久的心流感中起着至关重要的作用，因为一个人的注意力参与了任务可能产生的情绪类型。当手头的任务很简单时，人会感到冷漠或无聊，因为他们的注意力从任务上转移了。相反，当一个人全神贯注于任务但任务非常复杂时，他们会感到焦虑。也就是说，挑战超出了他们的能力。在理想情况下，玩家全神贯注于任务，其难度水平适合玩家的能力，且玩家有控制感，那么他们就可以进入心流。随着一个人掌握了一项挑战，其技能会提高，因此挑战的难度必须随之增加，以保持其处于心流状态，否则任务将不再令人愉快。

图 1 更详细地展示了一个人在执行任务时可能经历的八种情绪。心流状态发生在手头的任务既令人兴奋、具有挑战性又是可实现的时候，处于控制感和唤醒感这两种情绪之间。当执行任务所需的挑战和技能超出玩家的平均水平时，这些感觉会增强。

一般任务中心流体验的描述与玩家沉浸在游戏中的时的体验相同，例如忘记时间[8]。Cruz 等人[13]回顾了先前在电子游戏背景下适应心流理论的尝试。他们描述了心流要素与游戏特征之间的不同映射，以促进以玩家为中心的游戏设计，并培养玩家体验中的心流。

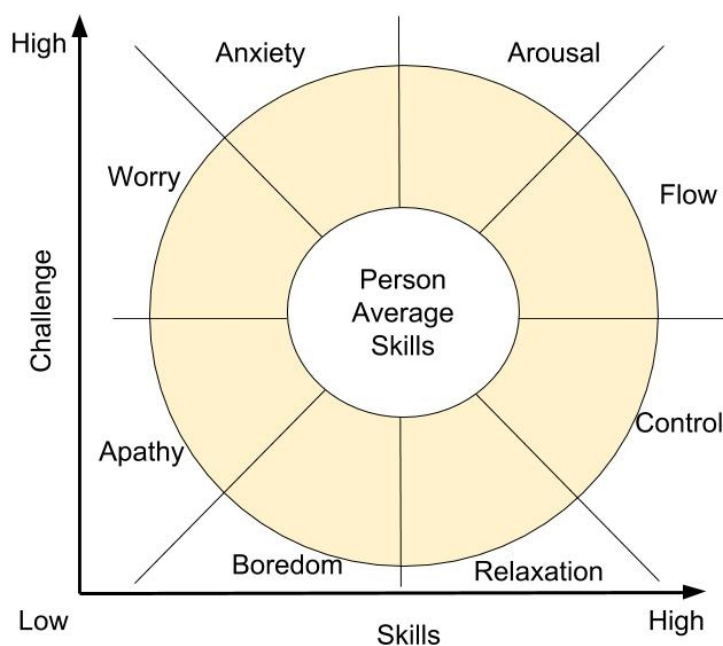


图 4 依据 Csikszentmihalyi 的“心流”模型[11]，心理状态在挑战水平和技能水平中的表现。

在建立心流理论与游戏设计之间的联系方面，GameFlow 模型[14]尤为相关。GameFlow 是一个玩家享受模型，由源自游戏用户体验文献的一组标准组成，并结构化为八个要素，这些要素可以映射到 Csikszentmihalyi 的心流概念。

2.2 心流与 DDA

大多数连接心流理论和 DDA 的尝试都使用监测生理变量（脑电图（EEG）、心率或皮肤电反应等）的设备来识别玩家的情绪状态并相应地调整游戏难度。Chanel 等人[15]描述了在三个不同难度水平下玩《俄罗斯方块》时识别情绪的早期尝试。Klasen 等人[16]评估了大脑对第一人称射击游戏中促成心流体验的不同维度的反应，表明挑战与技能的平衡、控制感和专注力最能代表心流体验。在[17]中描述了情绪检测在实际实施 DDA 系统中的首批应用之一，该系统利用玩家在玩节奏游戏时的 EEG 信号。在[18]和[19]中描述了针对射击游戏的更精细版本，作者使用不同的设备监测生理信号并调整游戏难度，以保持玩家的兴奋水平并改善游戏体验。

尽管最近像 BITalino 这样的项目降低了生物信号采集的准入门槛，但获取高质量生物信号的任务非常耗时，而且通常获取此类测量值的工具对用户来说并不舒适，需要将电极附着在身体的不同部位。因此，在不久的将来，玩家不太可能习惯定期使用生物信号采集硬件。作为生理变量的替代方案，心理学文献中可以找到许多自我报告问卷，这些问卷已被验证为评估心流的工具[20]。在这项工作中，我们建议使用事后问卷，而不是通过使用生理测量的情绪检测进行实时难度调整。

2.3 DDA 的其他方法

大多数 DDA 相关工作定义了针对不同类型游戏调整难度的特定技术。例如，[21]测试了五种不同的《俄罗斯方块》难度调整算法，这些算法不调整游戏速度，而是根据当前游戏状态通过选择方块来调整难度。调整非玩家角色（NPC）的 AI 是若干文献的焦点，例如[22]提出了一种调整敌人生成率和距离的算法，[23]针对 MOBA 游戏，或[24]和[21]针对赛车游戏。强化学习和进化计算等复杂技术通常应用于生成能够动态适应玩家的 NPC。本文所提工作的一个显著特点是关注“何时调整”，而非过多关注“如何调整”难度。

在[25]中，作者展示了开发通用 DDA 机制的初步努力，这是一种在线学习算法，

它将特定于游戏的修改难度的方式和当前玩家的游戏内历史作为输入，并产生适当的难度修改作为输出。然而，该论文仅涵盖了聚类成不同类型并根据简短的游戏玩法轨迹预测类型的初始步骤。同样针对玩家建模，[26]提出了一种与游戏无关的玩家表现预测方法。他们提出了一种数据驱动的张量分解方法，可以预测玩家技能掌握随时间的变化。然而，他们仅涵盖了玩家特征化的初始阶段。通常，难度调整阶段大多是特定于游戏的，根据游戏的不同可能或多或少费力。在这项工作中，我们根据玩家的技能水平以及当前游戏与先前好坏游戏的复杂性演变对比，来决定何时调整难度，这也将我们的工作与先前的玩家建模工作联系起来。

3 基于案例的 DDA

在本节中，我们介绍一种通用方法论，作为在电子游戏开发过程中实施 DDA 的指南，并解释我们如何在《俄罗斯方块》中实施其不同阶段。我们的目标是通过平衡玩家的真实技能与完成拟议挑战所需的技能，促进创建一个改善玩家体验的系统。

我们的方法论包含两个阶段（见图 2）。第一个阶段是“玩家预分析”，这是一个“离线”阶段，在此阶段我们收集不同玩家玩的一组游戏，然后分析这些游戏以了解不同玩家如何与游戏互动，以及哪些游戏变量有助于描述玩家在游戏过程中的表现。这是一个探索性阶段，将帮助我们创建和注释 DDA 系统使用的数据。第二个阶段是“在线”的，即在游戏过程中执行，由两个主要组件组成：玩家分类和难度调整，它们在游戏过程中连续执行。玩家分类组件将在游戏过程中通过将当前玩家的表现与其他先前玩家的表现进行比较来预测其技能水平。反过来，难度调整组件将决定是否以及如何调整游戏难度以适应当前玩家的技能水平。

请注意，这些阶段非常通用，可以使用不同的技术来实现。此外，在每个游戏中用于建模玩家表现和改变难度的特征将是特定于该游戏的。因此，游戏设计师在我们的方法论中扮演着重要角色，应参与识别正确的游戏变量，以建模玩家表现并动态调整游戏难度。

在接下来的章节中，我们将详细解释如何在《俄罗斯方块》中实施该方法论。我们使用基于案例的推理（CBR）方法[10]，其中案例代表先前游戏在短时间窗口内的

演变，我们关注两个被认为对在电子游戏中诱导心流特别相关的维度：控制感和挑战与技能的平衡。

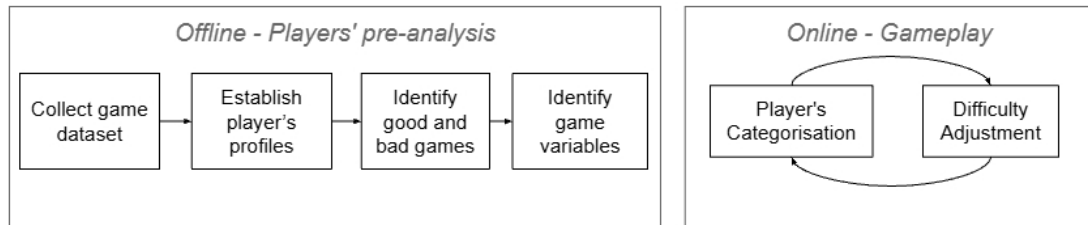


图 5 DDA 方法论框架。离线阶段的目标是收集、分析并标注一组游戏数据，这些数据将在第二阶段使用。

在线阶段由两个组成部分构成，持续分析玩家在游戏中的表现，并根据表现调整游戏难度。

3.1 玩家预分析

此阶段的目标是收集一组游戏轨迹，并使用不同的统计和可视化技术对其进行分析，以了解不同玩家如何与游戏互动，区分对玩家而言较满意和较不满意的游戏，并识别能更好地评估玩家在游戏过程中表现的游戏变量。

1) 收集游戏数据集：

此步骤的目标是收集足够的游戏轨迹及对这些游戏的评估，以代表潜在的玩家群体。在没有版主监督、玩家在不同时间参与的异步设置中，选择一种非常快速的方式在每局游戏后评估玩家满意度非常重要，因为尽管人们愿意玩几局《俄罗斯方块》，但他们不太愿意在每局游戏后填写调查问卷。任何耗时超过几秒钟的事情都可能阻止参与者玩下一局游戏。另一方面，同步设置成本更高，通常涉及较少的参与者，但版主的存在允许使用更复杂的满意度调查，并收集有关每个玩家和游戏的更多数据。

我们使用了异步设置，并通过向我们大学的不同邮件列表发送电子邮件来接触潜在参与者。目标参与者是来自不同学院（科学、工程、文学……）且具有不同电子游戏经验的 20-30 岁研究生。我们请他们协助我们进行一项关于电子游戏的研究，他们只需玩《俄罗斯方块》并回答一个简单的问题。他们没有受到任何其他方式的激励。他们玩的是一个名为“Tetris Analytics”的特殊版本，看起来像普通的《俄罗斯方块》游戏，但会将游戏轨迹存储在文件中，以便以后使用不同的可视化工具和机器学习算法进行重现和分析。参与者想玩多少局都可以。游戏中出现的新方块是随机选择的，并且只有一个标准难度等级。在每局游戏后，系统要求玩家使用 5 点李克特量表评估句子“这是一局好的俄罗斯方块游戏”来评估他们的体验，其中 1 表示非常

不同意，5 表示非常同意。我们收集了由 51 名不同玩家玩的 156 个游戏轨迹。9 局游戏评分为 1，29 局为 2，28 局为 3，55 局为 4，35 局评分为 5。

2) 建立玩家档案：

在一些游戏（特别是《俄罗斯方块》）中，有一个最终得分来衡量玩家在游戏过程中的表现。我们可以利用这个变量来标记玩家的档案或技能水平，但前提是游戏已经结束。此外，每个档案对应的分数范围或应该区分多少个档案并不明显。幸运的是，我们可以利用上一步收集的数据做出合理的选择。我们使用 K-均值（k-means）对最终得分进行聚类，并使用肘部法则（elbow method）选择了 3 个代表不同档案的聚类，我们将这些档案分别标识为：新手（0-2999 分，72 局），普通玩家（3000-5999 分，82 局）和专家（超过 6000 分，2 局）。然后，每局游戏都被注释了一个档案。

请注意，我们可以根据所需的粒度创建更多或更少的档案。我们可以使用对游戏设计师有意义的档案数量，或使用某种算法方法来选择最佳聚类数。

3) 识别好游戏和坏游戏：

在收集游戏数据集期间，每局游戏都使用 5 点李克特量表进行了满意度评分。如果是 4 或 5 分，我们认为该游戏是“好”的；如果是 1 或 2 分，则认为是“坏”的。3 分意味着玩家觉得游戏并非特别好或特别坏，因此我们在此分类中不考虑它们。此外，“坏”游戏根据其最终得分相对于同一档案玩家分数的比较，被进一步分为两个子类：太难和太容易。例如，如果一名普通玩家对某局游戏给出了糟糕的满意度评分，且该游戏的最终得分远低于普通玩家的平均得分，我们假设该游戏太难，从而对玩家体验产生了负面影响。数据集中有 90 局游戏被注释为好游戏，19 局为太难，4 局为太容易。我们将稍后使用这 3 个游戏子集来调整游戏难度，使其变得类似于好游戏，而不同于坏游戏。

4) 识别相关游戏变量：

此步骤的目的是选择一些代表玩家在游戏过程中表现的变量，并确定应该在游戏的哪些相关时刻测量它们。我们将通过描述游戏演变的时间序列来表示这些变量的演变。然后，我们将这些时间序列划分为表现窗口（performance windows），代表短时间内游戏的演变。这样，我们将能够通过比较当前玩家的最后一个表现窗口与我们数据集中的注释表现窗口来调整新游戏中的难度。

在《俄罗斯方块》中，我们区分两类决策。当一个方块出现在面板顶部时，玩家必须分析当前面板并决定希望方块最终落在何处。我们称这些为战术决策（tactical decisions），因为它们导致玩家希望在不久的将来达到的状态。然后，为了达到这些状态，玩家必须决定将当前方块从面板顶部带到期望最终位置所需的所有中间动作（平移和旋转）。本文我们将关注战术决策，因为我们认为它们代表了游戏中最显著的时刻，并且我们仅在每个方块落定在其最终位置时测量游戏的演变。我们将考虑导致最终位置的中间移动留待未来工作，这些移动无疑包含相关信息，但也更底层且噪声更大（例如，紧张的玩家会多次来回改变方块的旋转）。

选择用于在游戏过程中预测当前玩家档案的变量是一个试错过程，必须在游戏设计师的帮助下完成。基于我们对《俄罗斯方块》的了解，我们选择了以下变量来代表游戏的演变：

①方块编号：从游戏开始算起。由于我们只考虑战术决策，且每个方块对应一个战术决策，因此它代表时间。

②得分：当前方块落定后玩家累积的分数。

③面板高度：或方块占据的最高行。

④空洞：被另一个实心单元格完全覆盖的空单元格。

图 3 展示了这些游戏变量在一名普通玩家的游戏是如何演变的。该玩家在游戏结束前能够放置 64 个方块，总得分为 2480。黄线描述了得分的演变，随着每个新方块的放置略有增长，并在玩家消除一行时更大幅度增长。红线代表面板高度，当达到 20 时游戏结束。我们可以看到，在前 30 个方块期间，玩家在面板的下半部分进行游戏，但随后面板高度上升，因为方块下落太快，玩家无法紧凑地放置它们。同样，蓝线代表空洞的数量。

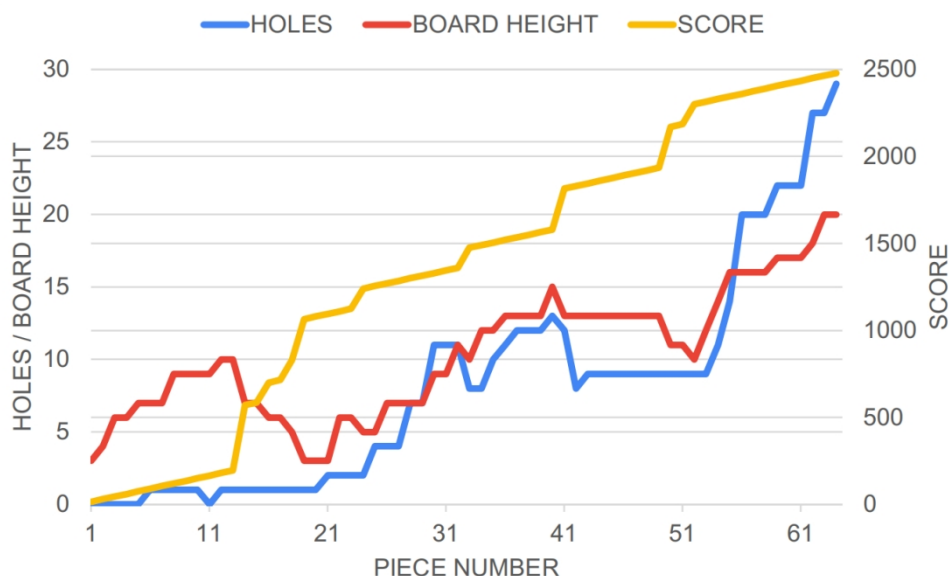


图 6 初学者玩家的游戏变量演变

每个游戏变量的演变可以表示为一个时间序列，其中每个值对应于连续时刻该变量的值。由于我们感兴趣测量玩家在游戏不同时刻的表现，我们将时间序列划分为表现窗口，代表短时间内所有变量的演变。这样，每个表现窗口是一个 $m \times n$ 的矩阵，其中 m 是变量的数量， n 是时间窗口的长度。

3.2 玩家分类

我们使用基于案例推理（CBR）方法来在游戏过程中决定新玩家的技能档案。每个案例的描述包含一个长度为 $n = 10$ 的表现窗口（描述 10 个方块期间游戏变量演变的时间序列），案例的解决方案包含玩该游戏的玩家的档案（根据其最终得分计算得出）。当玩家开始新游戏时，我们等待一段时间来构建她的第一个表现窗口，然后将其作为查询来检索案例库中 $k = 10$ 个最相似的案例。我们通过检索案例中的多数投票来预测当前玩家的档案。接下来，我们概述我们的方法，有关此组件的更详细描述请参见[27]。

当我们检索与当前表现窗口最相似的案例时，我们仅考虑在先前游戏中同一时间（方块编号）收集的案例，因为在游戏的不同时刻，两个非常相似的面板可能对玩家的技能水平有截然不同的含义。例如，一个只有前几行有方块且有少量空洞的游戏面板在游戏开始时是正常的，但只有专家级玩家才能在游戏进行到很晚且方块下落速度极快时保持这种面板配置。查询与案例之间的相似度计算为它们时间序列之间相似度

的线性组合：

$$\begin{aligned} sim_c(c_1, c_2) = & \alpha_1 sim_{ts}(c_1.score, c_2.score) + \alpha_2 sim_{ts}(c_1.holes, c_2.holes) \\ & + \alpha_3 sim_{ts}(c_1.height, c_2.height) \end{aligned}$$

其中权重 α 可以调整，以赋予每个时间序列更多或更少的重要性。在《俄罗斯方块》中，预测玩家技能水平最重要的变量是游戏得分（ $\alpha_1 = 0.70$ ），其次是空洞数量（ $\alpha_2 = 0.25$ ），最后是面板高度（ $\alpha_3 = 0.05$ ）。这些最佳权重是通过网格搜索计算得出的，增量为 0.05，并使用 10 折交叉验证来提高分类器的准确性。

我们使用基于欧几里得距离的相似度来计算 2 个时间序列之间的相似度[28]：

$$sim_{ts}(r, s) = 1 - \sqrt{\sum_{i=1}^n (r_i - s_i)^2}$$

其中 r 和 s 是大小为 n 的时间序列。

最后，游戏过程中不一致的玩家行为可能会导致档案预测频繁变化，从而导致游戏难度不断变化。例如，专家级玩家可能会在游戏某一点分心，表现得像新手，或者新手玩家可能会在一段时间内表现良好。游戏难度的突然变化通常会被认为是“有些奇怪的事情正在发生”，并引起玩家的不信任态度。此外，该模块的档案预测可能会失败（随着游戏的进行，预测会变得更加准确），因此建议使用某种惯性函数来避免在极端档案之间转换。在我们的实现中，我们使用了一种仅允许在相邻档案之间转换的惯性函数。也就是说，如果玩家在之前的表现窗口中被归类为新手，那么在当前的表现窗口中她只能被归类为新手或中级，无论她放置最后几个方块的表现有多好。

3.3 难度调整

此组件的目的是决定何时以及如何修改游戏难度。我们关注代表心流最佳心理状态的 9 个维度中的 2 个：挑战与技能的平衡和控制感。这样，是否修改游戏难度以及以何种强度修改的决定取决于玩家的档案以及她是否在其档案范围内处于控制状态。

1) 玩家是否处于控制状态？

控制感表明个人是否感觉有能力执行任务，因为任务处于适当的难度水平，这已被确定为心流体验中最重要的组成部分之一[11]。如果任务太难，主体会感到焦虑；如果太容易，主体会感到无聊，因此控制感介于这两个极端之间。为了间接测量玩家的控制感，我们计算游戏状态的复杂性，然后将其与我们标记的游戏集合中与当前玩

家处于同一档案的玩家的好游戏、太容易的游戏和太难的游戏的平均复杂性进行比较。为了让玩家处于控制状态，即执行具有适当难度水平的任务，上一个表现窗口期间的面板复杂性应该更类似于好游戏的平均复杂性，而不是太容易或太难的游戏。与前一个组件一样，我们只比较不同游戏中同一时刻（方块编号）的表现窗口的复杂性。

我们在每个方块落定在其最终位置后，使用以下公式计算游戏状态复杂性 C ：

$$C = a^2 + \sigma(\text{contour}) + h$$

其中 a 是面板上列的中位数高度的平方， $\sigma(\text{contour})$ 是列高度的标准差， h 是方块之间的空洞数。这些变量代表了随着下落方块速度增加玩家必须克服的挑战：低面板高度、均匀的面板轮廓以及方块之间很少的空洞。

图 4 显示了新手档案玩家随着游戏进行平均游戏状态复杂性及其标准差。红线、绿线和蓝线分别代表太难、好和太容易游戏的平均复杂性。随着游戏的进行，游戏的复杂性逐渐分开，并且每种游戏的复杂性以不同的速度增加。这种不同的轨迹也存在于普通玩家中，但线条相距不那么远。

最后，为了决定玩家是否处于控制状态（一个二元变量），我们将上一个表现窗口中每个游戏状态的复杂性与同一时刻的好游戏、太容易游戏和太难游戏的平均复杂性进行比较。当大多数这些复杂性更接近好游戏而不是太容易或太难的游戏时，玩家将被视为处于控制状态。如果玩家未处于控制状态，则游戏难度不适合她当前的技能水平。

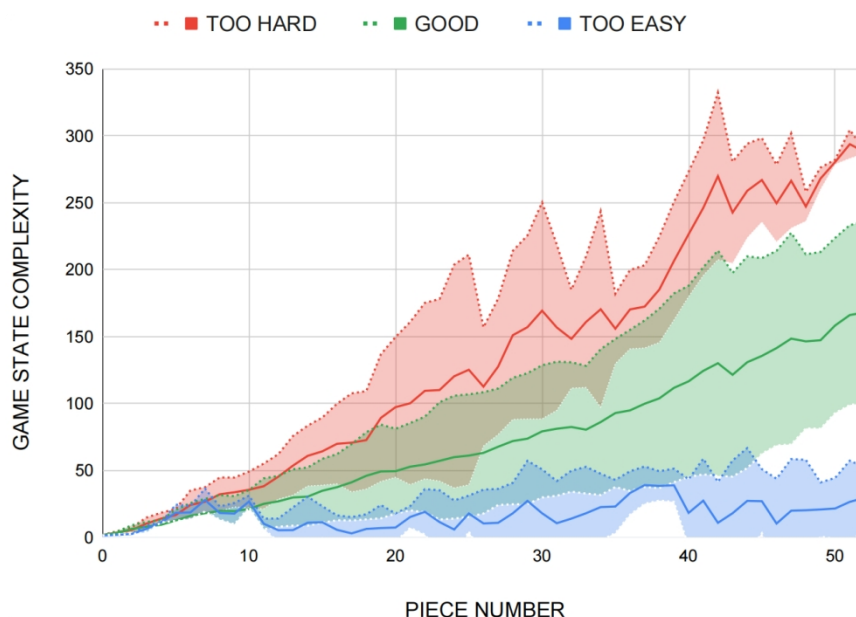


图 7 随游戏进展，初学者玩家的平均游戏状态复杂度及其标准差。

2) 修改《俄罗斯方块》中的难度

《俄罗斯方块》中的难度基本上取决于两个变量：交付的方块类型和下落速度。利用这两个变量，我们可以使游戏变得更难或更容易，并使其适应当前玩家的技能水平。

下一个方块可能是《俄罗斯方块》游戏中最重要的事件。一个适合当前面板配置的好方块可能会降低面板高度，而一个坏方块可能会大大增加面板的复杂性。我们使用启发式函数评估每个方块对当前面板的好坏程度，该函数评估将该方块放置在每个可能的最终位置所能达到的每个游戏面板。与每个游戏面板关联的启发式值是面板高度加上空洞数，因此值越小，面板越好。每个方块的启发式值是可达面板的最佳（最低）值（因此我们假设玩家将以最佳方式放置它）。值得一提的是，总是提供最好或最坏的下一个方块可能不是一个好主意，因为其中一些可能会被选择得太频繁而被玩家消极感知。最好是随机选择最好或最坏的方块之一，以确保足够的可变性。

游戏难度也可以通过改变方块下落的速度来改变。但我们必须小心行事，因为在《俄罗斯方块》中下落速度本应总是增加的，我们不想让玩家察觉到我们在改变游戏。此外，如果我们过多地降低下落速度，游戏可能永远不会结束。因此，在这种情况下，正确的方法是在每次玩家放置另一个方块时，稍微增加或减少下落速度的增量。

尽管我们的方法可用于使游戏变得更难或更容易，但在接下来描述的实验中，我们仅专注于使游戏更容易，从而帮助经验不足的玩家。我们认为新手玩家在享受游戏和进入心流方面有更多困难，而专家玩家已经具备实现拟议挑战所需的技能，几乎不需要帮助就能获得令人满意的体验。因此，我们定义了两个不同的适应级别，以产生考虑玩家档案和控制感的挑战-技能平衡：

① 速度-方块适应：此级别修改下落速度和下一个方块。下一个方块是在最好的 3 个候选者中随机选择的。下落速度率的增加量是正常情况的一半。当玩家是新手且未处于控制状态时，此适应级别处于激活状态。

② 仅方块适应：此级别仅修改下一个方块。当玩家是新手且处于控制状态，或者当玩家是普通玩家且未处于控制状态时，此适应级别处于激活状态。

4 实验

在实验中，我们比较了《俄罗斯方块》的三个不同版本：普通版（normal）、简单版（trivial）和平衡版（balanced, DDA）。普通版对应于原始的 Tetris Analytics 系统，其中下一个方块是随机选择的，而其他两个版本是同一系统的修改版。普通版是没有任何修改的原始《俄罗斯方块》游戏。简单版总是返回一个好的下一个方块，以便玩家可以轻松消除行。平衡版分析玩家的表现并相应地改变难度级别，如上一节所述。本实验的目的是测量玩家在游戏过程中体验到的心流。

实验将测试以下假设：

H1：使用 DDA（平衡版），我们能让更多玩家体验到心流，相比于不使用它（普通版）。

H2：与中级或高级玩家相比，新手更难体验到心流（普通版）。

H3：如果我们为低技能水平的玩家提供简单的《俄罗斯方块》版本，那么他们将体验到心流（简单版）。

H4：如果《俄罗斯方块》太容易，那么玩家将不会体验到心流（简单版）。

4.1 实验设置

实验是在现场进行的，有一名版主在场提供初步指导并引导实验。40 名志愿者参与了实验（除参与第三节 A1 部分的学生外的计算机科学学生）。他们被要求填写一份包含基本人口统计信息（年龄、性别）的表格，并回答一些关于他们使用电子游戏的问题：你是否是常客玩家？（是/否），你每周玩多少小时？（0-5, 5-10, 10-15, 15-20, +20），你知道《俄罗斯方块》吗？（是/否），你在《俄罗斯方块》中的技能水平如何？（从未玩过、新手、中级、高级）。

每位参与者首先玩 40 秒《俄罗斯方块》以熟悉控制并回忆游戏玩法。接下来，每位参与者玩 3 局完整的游戏，并在每局游戏后填写简版心流状态量表（SHORT Flow State Scale, S-FSS）[12] 问卷。最后，每位参与者选择她最喜欢的游戏（1、2 或 3）。实验设计时长约为 30 分钟。

参与者不知道实验的目的，也不知道这些游戏对应于《俄罗斯方块》的不同版本。参与者被随机分为 3 组，以便他们按不同顺序玩不同版本：

组 1：简单版，平衡版，普通版。

组 2：普通版，简单版，平衡版。

组 3：平衡版，普通版，简单版。

简版心流状态量表（S-FSS）[12]确定玩家是否在活动中进入了心流（表 I）。它包含九个项目，每个项目的得分代表心流的一个维度。该量表设计为心流的事后评估，说明旨在将受试者与最近完成的活动联系起来。当量表在游戏结束附近实施时，可以对心流状态进行更准确的评估。回答范围从 1（非常不同意）到 5（非常同意）。较低的回答值表明受试者的体验本质上不属于心流。相反，较高的回答值象征着个人实质性地体验了心流。状态量表上的平均分 3 代表选择“既不同意也不反对”。这个平均分可能表示某种程度的认可，但也可能意味着关于该项目与个人体验相关性的某种模糊性。然而，将中等水平的分数解释为既不是个人经历心流的有力证据，也不是个人经历不包含心流的有力证据是合理的。

表 I 短期心流状态量表中的每个项目及其代表的维度[12]

#	维度	句子
1	挑战-技能平衡	我觉得自己足够有能力应对当时情境的要求。
2	行动与意识的融合	我做事时非常自发、自动，完全不需要思考。
3	清晰的目标	我对自己想做的事情有很明确的认识。
4	明确的反馈	在进行任务/活动时，我很清楚自己做得怎么样。
5	集中注意力	我完全专注于当前的任务。
6	控制感	我有完全控制自己所做事情的感觉。
7	自我意识的丧失	我不担心别人对我的看法。
8	时间的变换	时间的流逝似乎和正常情况有所不同。
9	内在奖励体验	我觉得这个经历极具奖励性。

4.2 结果

40 名参与者的年龄在 18 至 42 岁之间，平均年龄为 27.3 岁。超过一半的参与者每周玩电子游戏超过 5 小时（见图 5）。此外，超过一半的参与者认为自己是新手（见图 6）。

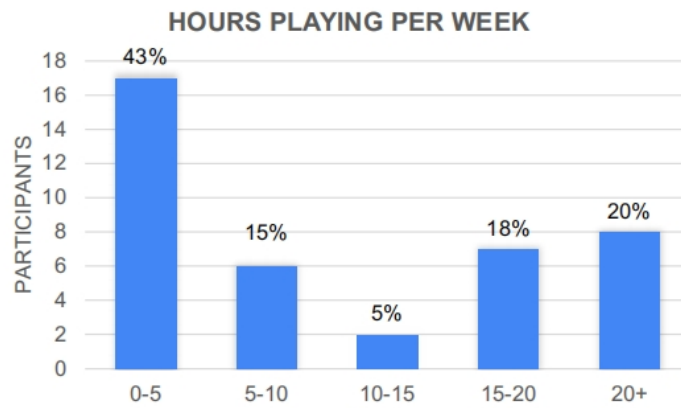


图 8 参与者每周玩视频游戏的小时数

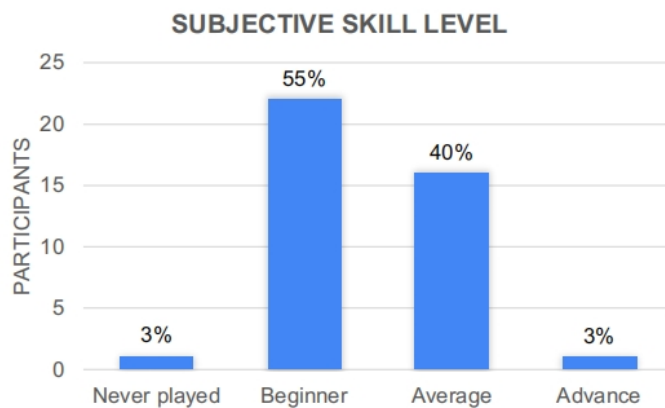


图 9 参与者的主观技能水平

让我们从参与者最喜欢的《俄罗斯方块》版本开始分析。在实验的最后一个问题中，参与者选择了 3 个游戏中他们最喜欢的一个。图 7 显示了每个版本被选择的百分比：20% 的玩家选择了普通版，27.5% 选择了简单版，52.5% 选择了平衡版。如果我们根据玩家的主观《俄罗斯方块》技能水平（他们在初始问卷中认为自己拥有的水平）过滤结果，新手和中级玩家选择平衡版的频率都是下一个版本（简单版）的两倍。高级玩家也选择了平衡版，而从未玩过《俄罗斯方块》的玩家选择了简单版。

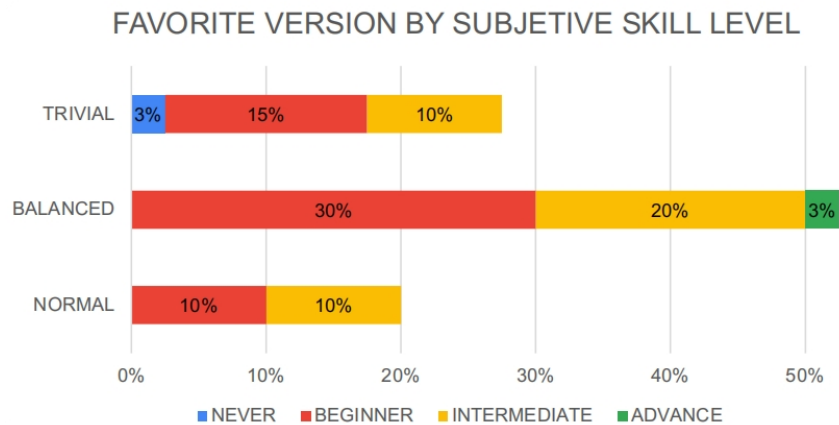


图 10 按主观技能水平过滤的最喜欢的版本

实验的主要目标是确定哪个版本的《俄罗斯方块》能产生最高的心流状态。由于有 40 名参与者，且每人都玩了 3 个版本的《俄罗斯方块》，因此我们每个版本有 40 份简版心流状态量表问卷。为了确定某个版本的《俄罗斯方块》是否产生比其他版本统计上显著更高的心流状态，我们成对比较《俄罗斯方块》版本，对每一对考虑的版本运行配对 t 检验：普通版 vs 平衡版；平衡版 vs 简单版；普通版 vs 简单版。由于我们对同一数据集进行了 3 次不同的比较，我们应用邦弗朗尼校正 (Bonferroni correction) [29]，不是考虑 5% 的第一类错误率 [30]，而是使用 0.017% ($\approx 0.05/3$) 的显著性阈值。

如表 II 所示，平衡版的平均心流得分 ($M=4$, $SE=0.602$) 显著高于普通版的平均心流得分 ($M=3.778$, $SE=0.54$)， $p < .017$ 。因此，我们接受 H1：使用 DDA 我们能让更多玩家体验到心流。

表 II 各版本《俄罗斯方块》的平均心流状态及 p 值

	平均心流			p 值		
	普通版	平衡版	简单版	普通版 vs 平衡版	平衡版 vs 简单版	普通版 vs 简单版
心流得分	3.778	4.000	4.003	0.001	0.973	0.033

对于测试 H2：与中级或高级玩家相比，新手更难体验到心流，我们根据参与者在普通版《俄罗斯方块》游戏中的最终得分将他们按实际技能水平分组（与之前使用的主观技能水平相对）。21 名参与者被归类为新手，19 名为中级玩家。我们发现两组玩家的平均心流得分没有显著差异，因此我们拒绝 H2。然而，如果我们分离数据并分别对新手（表 III）和中级玩家（表 IV）再次运行配对 t 检验，我们会得到额外的见解。对于新手，平衡版和简单版提供的心流都显著高于普通版 ($p < .017$)，而对于中级玩家，只有平衡版是这种情况。

对于测试 H3：如果我们为低技能水平的玩家提供简单的《俄罗斯方块》版本，那么他们将体验到心流，我们在新手玩家组内比较了简单版和普通版。平衡版和简单版都比普通版容易。平衡版在必要时调整挑战使其更容易，而简单版总是提供帮助。简单版的平均心流得分 ($M=4.037$, $SE=0.628$) 显著高于普通版的平均心流得分 ($M=3.778$, $SE=0.415$)， $p < .017$ 。这些结果表明，两个提供更容易挑战的版本都帮

助新手比在普通版中更强烈地体验到心流。因此，我们接受 H3。

最后，我们接受 H4：如果《俄罗斯方块》太容易，那么玩家将不会体验到心流，因为在我们的实验中，简单版中产生的心流并不显著高于 ($p < .017$) 普通版或平衡版测量到的心流（见表 II）。然而，如果仅考虑新手（表 III），那么简单版中的心流显著高于普通版，因此我们对于新手可以拒绝 H4。

表 III 新手玩家各版本平均心流得分及 p 值

	平均心流			p 值		
	普通版	平衡版	简单版	普通版 vs 平衡版	平衡版 vs 简单版	普通版 vs 简单版
心流得分	3.788	3.979	4.037	0.010	0.295	0.004

表 IV 中级玩家各版本平均心流得分及 p 值

	平均心流			p 值		
	普通版	平衡版	简单版	普通版 vs 平衡版	平衡版 vs 简单版	普通版 vs 简单版
心流得分	3.766	4.023	3.965	0.003	0.689	0.159

4.3 讨论

根据我们在本次实验中的结果，我们可以得出结论：

- ① 玩平衡版时的平均心流体验显著高于玩普通版时。
- ② 平衡版的平均心流体验并不显著高于简单版。
- ③ 对于新手，简单版和平衡版相比普通版都改善了心流体验。
- ④ 我们没有证据表明新手比中级和高级玩家更难进入心流。

我们原以为与平衡版相比，玩家会觉得简单版无聊，但事实并非如此。这可能是由于大多数实验参与者认为自己是新手，并且很长时间没有玩过《俄罗斯方块》，所以他们很高兴玩一个简单的游戏版本。此外，参与者在简单版上的得分（平均分 4720）高于平衡版（平均分 3659），这可能会影响问卷的回答。此外，每局游戏持续约 3-5 分钟，即使游戏非常简单，这个时长也可能不足以让人感到无聊。我们相信，如果玩家必须玩几分钟以上，他们会发现简单版很乏味。此外，53% 的参与者选择平衡版作

为最喜欢的版本，因此我们可以得出结论，平衡版《俄罗斯方块》提供了比简单版更有价值的体验。

关于研究的局限性，我们确定了一些未来应进一步调查的方面。首先，40 名参与者都与计算机领域相关，因此他们可能无法代表普通玩家群体。其次，我们的研究没有考虑导致某些人以更大或更小的困难和强度体验心流的个人特征。第三，我们的研究侧重于新手或中级玩家，我们在实验中使用的平衡版只降低游戏难度，从不增加难度。动态难度调整如何影响需要更大挑战的专家玩家仍有待研究。最后，我们的实验只考虑了非常短的游戏会话。研究更长的时间段会很有趣，看看游戏难度是否随着玩家的进步而演变。

5 结论与未来工作

本文提出了一种新颖的方法论，作为利用心流理论思想在电子游戏中实施 DDA 的指南。我们的方法论分为两个阶段：预分析和在线游戏。在预分析期间，我们收集并分析游戏轨迹，以识别最相关的游戏变量来评估玩家的表现。我们还创建了一个标记有玩家技能水平和满意度分数的案例库。第二阶段发生在游戏过程中，持续预测当前玩家的档案，并决定何时以及如何调整游戏难度。为了预测玩家的档案，我们将她在上一个表现窗口期间的表现与存储在案例库中的表现进行比较。反过来，是否改变游戏难度的决定取决于玩家是否处于控制状态，这是通过将最后游戏状态的复杂性与案例库中被具有相同档案的其他玩家评为好、太容易或太难的游戏的平均复杂性进行比较来间接测量的。

我们在《俄罗斯方块》中实施了这种方法，并进行了一项实验，结果表明使用 DDA 的平均心流体验显著高于普通版。我们发现 DDA 和简单版之间的心流体验总体上没有显著差异。我们认为这是因为参与者大多是几乎没有玩过《俄罗斯方块》经验的新手，而且只玩了一局持续 3 或 4 分钟的简单游戏，这不足以让他们感到乏味。然而，如果我们分别研究新手和中级玩家，就可以发现简单版和平衡版之间的差异。虽然我们发现对于新手来说，这两个版本相对于普通版都有显著差异，但对于中级玩家，我们只发现平衡版优于普通版。

尽管这项工作侧重于特定的游戏，但我们相信所提出的方法论足够通用，可以应用于广泛的游戏。在特定游戏中使用此方法论必须做出的主要决定是：（1）识别最相关的游戏变量以评估玩家在游戏过程中的表现，（2）决定用于 DDA 的表现窗口的长度，（3）定义一个函数来测量表现窗口期间的游戏复杂性，以及（4）决定如何根据玩家在任何给定时间的需求使游戏变得更难或更容易。例如，在像《太空侵略者》（Space Invaders）这样的经典游戏中，玩家的表现可以使用诸如游戏得分、摧毁的飞船和外星人数、未击中任何敌人的射击数量等变量来衡量……；新的表现窗口可以每 10、20 或 30 秒创建一次，具体取决于我们希望多久重新评估一次游戏难度，每隔几秒收集一次值；游戏的复杂性可以使用外星人的速度、它们到地面的距离、剩余供玩家掩护的障碍物数量等来衡量……；游戏难度可以通过改变外星人的速度或其射击频率来改变。当然，这些只是想法，应该由既详细了解不同游戏机制的游戏设计师，也由能够发现和验证游戏轨迹中模式的数据分析师进行测试和评估。

我们的提议也有一些局限性。我们使用基于案例的实现，因此我们需要一个能够有效代表不同类型玩家和游戏（具有良好的覆盖率和案例多样性）的良好案例库。此外，决定使用事后问卷而不是实时传感器来测量玩家体验具有一些优势（这是一种非侵入式方法，不会改变玩家的体验），但也意味着我们只能获得关于整个游戏（或至少是游戏的一个关卡）的反馈，因此我们无法区分哪些具体部分在玩家中诱导了更多或更少的心流。作为未来工作的一部分，我们希望研究使用非侵入式设备来识别和标记案例库中的好、太容易和太难的游戏片段。请注意，这些设备仅在构建案例库期间需要，而在游戏发布后执行 DDA 时不需要。近年来智能手表和手环的惊人进步让我们在这方面感到充满希望。事实上，使用这些设备还可以通过检查玩家随后的生物识别反应来帮助我们测量难度调整的有效性。

尽管本文提出的方法论是通用的，但我们仅在新手和中级玩家中测试了它，使游戏变得更容易，而从未更难。我们还希望在更长的时间段内测试这种类型的 DDA，看看游戏难度是否随着玩家的进步而演变。我们对在严肃游戏中使用 DDA 感兴趣，在这类游戏中，保持玩家参与对于在较长时间内增强学习至关重要。

致谢

这项工作得到了西班牙科学与创新委员会（PID2020-114596RB-C21）和马德里康普顿斯大学（Group 921330）的部分支持。我们感谢匿名审稿人提出的有益和深刻的评论。

附录 B 外文原文

IEEE TRANSACTIONS ON GAMES, VOL. XX, NO. X, MONTH YEAR

1

Measuring Control to Dynamically Induce Flow in Tetris

Diana Sofía Lora-Ariza, Antonio A. Sánchez-Ruiz, Pedro Antonio González-Calero, and Irene Camps-Ortueta

Abstract—Dynamic Difficulty Adjustment (DDA) is a set of techniques that aim to automatically adapt the difficulty of a video game based on the player's performance. This paper presents a methodology for DDA using ideas from the theory of flow and case-based reasoning (CBR). In essence we are looking to generate game sessions with a similar difficulty evolution to previous game sessions that have produced flow in players with a similar skill level. We propose a CBR approach to dynamically assess the player's skill level and adapt the difficulty of the game based on the relative complexity of the last game states.

We develop a DDA system for Tetris using this methodology and show, in a experiment with 46 participants, that the DDA version has a measurable impact on the perceived flow using validated questionnaires.

Index Terms—Dynamic Difficulty Adjustment, Flow, Artificial Intelligence, Video Games.

I. INTRODUCTION

THE video game industry has grown rapidly in recent years. In 2020, this industry generated revenues above \$170 billion and it is expected to continue growing in the following years. This growth has motivated investors to finance with additional resources the development of new titles, and game studios to look for new ways to overcome the inherent difficulties involved in creating and maintaining good quality video games. The production of an average AAA video game takes between 1-3 years not just because the time required to create all the assets and game mechanics but because all these ideas must be tested continually to validate that they really work and the game is entertaining. This way, game designers play a fundamental role in the production of video games. They are usually in charge of defining the game levels, the puzzles, the controls, the game mechanics, the character dialogues, ... and especially the difficulty levels of the game that usually involve defining different sets of behaviors for the AI characters that will be used according to the player's skill level [1]. In other words, the "intelligence" seen in most video games today is the result of trying to anticipate different player behaviors and then, during the production stage, to implement a standard set of actions to respond appropriately in each expected scenario. Most of the content in video games today is predefined, created during the development process, with the hope that it will be adequate for most of the players, challenging but not too difficult, but that is not always achieved. In fact, player's feedback is an essential tool

for designers to distinguish between extremely difficult tasks and a great challenge [2].

An "intelligent" video game should be able to decide what to do in scenarios that the designers could not anticipate and provide an appropriate response [3]. In particular, the classical predefined difficulty levels present in most video games are not optimal for all players, and the difficulty of the game may not evolve at the same rate as the player's skills, making the game too easy or too difficult. In this sense, *difficulty* should not be understood as a static property, but a subjective factor derived from the interaction between the player and the proposed challenge [4]. Dynamic Difficulty Adjustment (DDA) is a set of techniques that aim to automatically adapt the difficulty of the game based on the player's performance [5]. That is, the player-game interactions are continuously evaluated to change the difficulty of the game according to the player's current needs with the goal of keeping the player engaged. A notable advantage of using DDA is the reduction in video game production costs, because if the game can adapt itself, then designers and developers require less effort trying to anticipate all possible situations [6]. For this reason, DDA has captured the attention of leading companies in the video game industry. Lopes and Bidarra [7] surveyed the state of art of adaptivity in games and simulations, from both academia and industry, and concluded that adaptivity is establishing itself as a rapidly maturing field and that current advances show good results in adapting towards an optimal challenge level and emotional states such as fun, frustration, predictability, anxiety or boredom.

Flow is a mental state in which a person performing some activity is fully immersed in a feeling of focus, full involvement, and enjoyment, usually resulting in a transformation in the sense of time. The connection between flow and immersion in video games is quite direct since the ability to transport its players to their personal flow zones has been identified as a requirement for a well-designed game [8].

This paper presents a novel methodology for DDA using ideas from the theory of flow [9] and case-based reasoning (CBR) [10]. A key advantage of using flow theory is that, since it is a very popular theory in Psychology research, there are different instruments to measure it, both on the basis of physiological measurements and posteriori questionnaires. Those instruments can be used to identify previous games that have produced flow in players. Our methodology aims to generate flow by altering the difficulty of the game so that its evolution is similar to previous games that have produced flow in players with a similar skill level. We use a CBR approach to dynamically assess the player's skill level and adapt the

The authors are with the Universidad Complutense de Madrid, Madrid 28040, and Universidad Francisco de Vitoria, Madrid 28223, Spain (email: dlora@ucm.es, antsanche@ucm.es, pagoncal@ucm.es, irene.camps@ufv.es). Manuscript received July 31, 2021; revised January 25, 2022.

difficulty of the game using a measure of control based on the relative complexity of the current game and a set of previous similar labeled games. We have implemented a DDA system for Tetris and show, using validated flow questionnaires, that has a measurable impact on the players.

The rest of the paper runs as follows. In next section we briefly introduce the theory of flow and presents its relation to DDA in games. Then, in Section III, we present the case-based DDA methodology exemplified with Tetris. Next, we describe an experiment that shows that our approach effectively improves the sense of flow in players in Section IV. The paper concludes with some conclusions and future lines of research.

II. RELATED WORK

A. The flow theory

Csikszentmihalyi [9] defines flow as *the optimal experience when nothing else matters*. Entering flow depends on creating the perfect balance between the perceived challenges of the task at hand and one's perceived abilities. Nakamura et al. [11] say that it is the subjective challenges and subjective skills, not the objective ones, that influence the quality of a person's experience. In other words, the person must have confidence in their capacity to complete the task successfully to enter flow. Csikszentmihalyi establishes nine dimensions which together represent the optimal psychological state of flow [12]: challenge-skill balance, merging action-awareness, clear goals, unambiguous feedback, concentration on the task at hand, sense of control, loss of self-consciousness, the transformation of time and autotelic experience.

Nakamura et al. [11] state that attention plays an essential role in achieving a lasting feeling of flow because a person's attention plays a part in the type of emotions that the task can generate. A person experiences apathy or boredom when the task at hand is simple, as their attention is away from the task. On the contrary, a person feels anxiety when they have their full attention on the task, but it is very complex. That is, the challenge exceeds their capacities. In an ideal situation, the player has full attention on the task, its difficulty level is correct to the player's capacities, and the player has a feeling of control, then they can enter flow. As a person masters a challenge, her skills increases and thus the challenge must grow in difficulty along with the person to keep them in flow or, otherwise, the task ceases to be enjoyable.

Figure 1 shows in more detail the eight emotions a person may experience while performing a task. The state of flow occurs when the task at hand is exciting and challenging but achievable, between the emotions of control and arousal. These feelings intensify when the challenges and skills required to perform the task are beyond the player's average levels.

Descriptions of the flow experience in general tasks are identical to those experienced by players when immersed in games, such as losing track of time [8]. Cruz et al. [13] review previous attempts to adapt the flow theory in the context of video games. They describe different mappings between flow elements and game features in order to promote player-centered game design, and foster flow in the player experience.

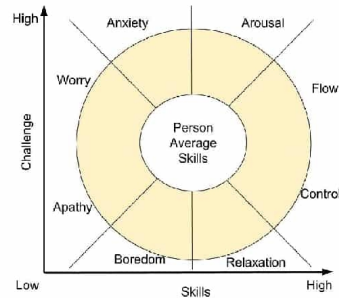


Fig. 1. Mental state in terms of challenge level and skill level, according to Csikszentmihalyi's flow model [11].

Of particular relevance in establishing the connection between flow theory and game design is the GameFlow model [14]. GameFlow is a model of player enjoyment, comprised of a set of criteria derived from games user experience literature and structured into eight elements that can be mapped to Csikszentmihalyi's concept of flow.

B. Flow and DDA

Most attempts to connect the flow theory and DDA make use of devices that monitor physiological variables (electroencephalograms (EEG), heart rate or galvanic skin response among others) to identify the player's emotional state and adjust the difficulty of the game accordingly. Chanel et al. [15] describe an early attempt on identifying emotions when playing Tetris at three different difficulty levels. Klasen et al. [16] assess how the brain responds to the different dimensions that contribute to the flow experience in a first-person shooter game, showing that the balance between challenge and skill, sense of control and concentration are most representative of the flow experience. One of the first applications of emotion detection to actually implement a DDA system is described in [17] using players' EEG signals during playing a rhythm game. More elaborated versions, both for shooter games, are described in [18] and [19] where the authors use different devices to monitor physiological signals and adapt the difficulty of the game to maintain the player's level of excitement and improve the gaming experience.

Although recent projects like BITalino have lowered the entry barrier of bio-signal acquisition, the task of obtaining bio-signals of good quality is time-consuming, and typically tools for obtaining such measures are not comfortable for the user, requiring electrodes attached to different parts of the body. For that reason, it is not foreseeable that gamers will become accustomed to using bio-signal acquisition hardware on a regular basis in the near future. As an alternative to physiological variables, numerous self-report questionnaires can be found in the literature of Psychology that have been validated as instruments for assessing flow [20]. In this work, we propose the use of ex-post questionnaires instead of real-

time difficulty adjustment through emotion detection using physiological measures.

C. Other approaches to DDA

Most of the related work in DDA defines specific techniques for adapting difficulty in different types of games. For example, [21] tested five different algorithms for adjusting difficulty in Tetris that, instead of adjusting game speed, adjust difficulty by choosing blocks based on the current game state. Adapting the AI of non-player characters (NPC) is the focus of several contributions, such as [22] that propose an algorithm for adjusting spawn rate and distance of the enemies, [23] in MOBA games, or [24] and [21] in racing games. Sophisticated techniques such as reinforcement learning and evolutionary computation are typically applied in the generation of NPC that can dynamically adapt to the player. One distinguishing feature of the work presented in this paper is the focus on "when to adapt" and not so much on "how to adapt" the difficulty.

In [25] the authors present an initial effort in developing a universal mechanism for DDA, an online learning algorithm that takes as input game-specific ways to modify difficulty and the current player's in-game history and produces as an output an appropriate difficulty modification. Nevertheless, the paper just covers the initial step of clustering into different types and predicting the type from short traces of gameplay. Also for player modeling, [26] presents a game-independent method for predicting player performance. They proposed a data-drive tensor factorization approach that can predict changes in players' skill mastery over time. Nevertheless, they only cover the initial stage of player characterization. In general, the difficulty adjustment phase is mostly game-specific and may be more or less laborious depending on the game. In this work, we decide when to adjust the difficulty based on the player's skill level and evolution of the complexity of the current game compared to other previous good and bad games, which also connects our work with previous work on player modeling.

III. CASE-BASED DDA

In this section we introduce a general methodology that serves as a guide for implementing DDA during the video game development process and explain how we have implemented its different phases in Tetris. Our goal is to facilitate the creation of a system to improve the player's experience by balancing her real skills and the skills required to complete the proposed challenges.

Our methodology consists of 2 phases (see Figure 2). The first one, *players' pre-analysis*, is an "offline" phase in which we collect a set of games played by different players and then we analyze those games to learn about how different players interact with the game and which game variables are useful to describe the player's performance during the game. This is an exploratory stage that will help us to create and annotate the data used by the DDA system. The second phase is "online" in the sense that it is performed during the game, and consists of 2 main components, *players' categorization* and *difficulty*

adjustment, that are executed one after the other continuously during the game. The *player categorization* component will predict the skill level of the current player *during the game* comparing her performance with the performance of other previous players. The *difficulty adjustment* component, in turn, will decide if and how to adapt the difficulty of the game to meet the current player's skill level.

Note that these phases are very general and can be implemented using different techniques. Moreover, the features used in each game to model the player's performance and to alter the difficulty will be specific for each game. For this reason, game designers play an important role in our methodology and should be involved to identify the right game variables to model the player's performance and dynamically adapt the difficulty of the game.

In the following sections we explain in detail how we have implemented this methodology in Tetris. We use a case-based reasoning approach (CBR) [10] in which cases represent the evolution of previous games during short time windows, and we focus on 2 flow dimensions that have been identified as especially relevant to induce flow in video games: sense of control and challenge-skill.

A. Players' pre-analysis

The goal of this phase is to collect a set of game traces and analyze them using different statistical and visualization techniques to understand how different players interact with the game, differentiate games that are more and less satisfactory for the players, and identify the game variables that better assess the performance of the player during the game.

1) *Collect game dataset*: The goal in this step is to collect enough game traces and evaluations of those games to represent the potential population of players. In asynchronous setups where players participate at different times without the supervision of some moderator, it is important to choose a very quick way to evaluate players' satisfaction after each game because, while people were willing to play a few games of Tetris, they were less willing to fill out a survey after each game. Anything that takes more than a few seconds may discourage the participants from playing another game. On the other hand, synchronous setups are more expensive and usually involve less participants, but the presence of a moderator allows to use more sophisticated satisfaction surveys and collect more data about each player and game.

We used an asynchronous setup and reached potential participants sending emails to different mailing lists in our university. The target participants were grad students in their 20-30s from different Faculties (sciences, engineering, literature, ...) and with different experience playing video games. We asked them to assist us with a research study about video games in which they only had to play Tetris and answer one simple question. They were not incentivized to participate in any other way. They played a special version of Tetris called Tetris Analytics that looks like a normal Tetris game but stores the game traces in files so that they can be reproduced and analyzed later using different visualization tools and machine learning algorithms. The participants could play as many

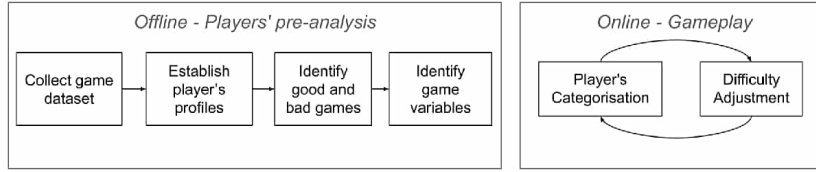


Fig. 2. DDA methodology scheme. The goal of the offline phase is to collect, analyze and annotate a set of games that will be used in the second phase. The online phase consists of 2 components that continuously analyze the player performance during the game and adapt the difficulty accordingly.

games as they wanted. The new pieces that appeared in the game were randomly selected and there was only one standard difficulty level. After each game, the system asked the players to assess their experience evaluating the sentence *It was a good Tetris game* with a 5-value Likert scale, where 1 means strongly disagrees and 5 strongly agrees. We collected 156 game traces played by 51 different players. 9 games were evaluated with a score of 1, 29 games with 2, 28 games with 3, 55 games with 4 and 35 games with a score of 5.

2) *Establish player's profiles*: In several games, and in Tetris in particular, there is a final score that measures the performance of the player during the game. We can use this variable to label player's profile or skill level but only when the game has ended. Also, it is not obvious what score ranges correspond to each profile or how many profiles should be differentiated. Fortunately, we can use the data collected in the previous step to make reasonable choices. We used k-means to cluster the final scores and the elbow method to select 3 clusters that represent different profiles that we identify with *beginners* (0-2999 points, 72 games), *average players* (3000-5999 points, 82 games) and *experts* (more than 6000 points, 2 games). Then, each game was annotated with a profile.

Note that we can create more or less profiles depending on the desired granularity. We can use a number of profiles that makes sense for the game designers or use some algorithmic approach to select an optimal number of clusters.

3) *Identify good and bad games*: During the collection of the game dataset, each game was evaluated with a satisfaction score using a 5-value Likert scale. We consider that a game was *good* if it was evaluated with a score of 4 or 5, and it was a *bad* if it was evaluated with a score of 1 or 2. A score of 3 means that the player did not feel the game was especially good or bad, so we do not consider them in this classification. Additionally, bad games are divided into two subcategories, *too hard* and *too easy*, depending on the final score of the game with respect to the scores of the players within the same profile. For example, if an average player evaluates a game with a bad satisfaction score and the final game score is much lower than the mean score of average players, we assume the game was too difficult and that produced a negative effect in the player's experience. 90 games from the dataset were annotated as good games, 19 as too hard and 4 as too easy. We will use these 3 subsets of games later to adjust the difficulty of the game so that it becomes similar to the good ones and different to the bad ones.

4) *Identify relevant game variables*: The purpose of this step is to select some variables that represent the player's performance during the game, and to identify the relevant moments of the game in which we should measure them. We will represent the evolution of these variables by means of time series describing the evolution of the game. We will then divide these time series into *performance windows* that represent the evolution of the game over short periods of time. This way, we will be able to adapt the difficulty in new games comparing the last performance window of the current player with the annotated performance windows in our dataset.

In Tetris we distinguish two types of decisions. When a piece appears at the top of the board, the player must analyze the current board and decide where she wants the piece to end up. We call these decisions tactical decisions, since they lead to states that the player wants to reach in the near future. Then, to reach those states, the player must decide all the intermediate actions (translations and rotations) needed to bring the current piece from the top of the board to the desired final position. In this paper we will focus on tactical decisions, since we think they represent the most significant moments of the game, and we will measure the evolution of the game only when each piece settles in its final position. We leave for future work to take into account also the intermediate moves that lead to the final position, which undoubtedly contain relevant information, but are also lower-level and noisier (e.g. nervous players change the rotation of the piece back and forth several times).

The selection of the variables that will be used to predict the current player profile *during* the game is a trial-and-error process that must be done with the help of the game designers. Based on our knowledge of Tetris we have selected the following variables to represent the evolution of the game:

- *Piece number* from the start of the game. As we only consider tactical decisions and there is one tactical decision per piece, it represents time.
- *Score* accumulated by the player once the current piece is settled.
- *Board height* or the highest row occupied by a piece.
- *Holes* or empty cells that are completely covered by another full cell.

Figure 3 shows how these game variables evolve in a game of an average player. This player was able to settle 64 pieces before the game finished and got a total score of 2480. The yellow line describes the score evolution that grows slightly



Fig. 3. Evolution of the game variables for a beginner player.

with each new piece and more abruptly when the player makes a new line. The red line represents the board height and when the game ends when it reaches 20. We can see that during the first 30 pieces the player played in the lower half of the board, but then the board height grew up because the pieces fell too quickly for the player to set them compactly. Similarly, the blue line represents the number of holes.

The evolution of each game variable can be represented as a time series in which each value corresponds to the value of the variable in consecutive instants of time. Since we are interested in measuring the player performance in different moments of the game, we divide the time series in *performance windows* that represent the evolution of all the variables during a short period of time. This way, each performance window is a matrix of $m \times n$ where m is the number of variables and n is the length of time window.

B. Player's Categorization

We use a case-based reasoning (CBR) approach to decide the skill profile of a new player during the game. The description of each case contains a performance window of length $n = 10$ (time series describing the evolution of the game variables during 10 pieces), and the solution of the case contains the profile of the player that played that game (computed from its final score). When a player starts a new game, we wait some time to construct her first performance window and then we use it as a query to retrieve the $k = 10$ most similar cases in the case base. We predict the profile of the current player by majority vote among the retrieved cases. Next, we resume our approach, see [27] for a more detailed description of this component.

When we retrieve the most similar cases to the current performance window, we only consider cases collected at the same time (piece number) in previous games because two very similar boards in different moments of the game can have very different implications regarding the player's skill level. For example, a game board with pieces only in the first rows and a few holes is normal at the beginning of the game but only expert players can have that same board configuration when the game is well advanced and the pieces fall with great speed.

The similarity between a query and a case is calculated as a linear combination of the similarities between their time series.

$$\begin{aligned} sim_c(c_1, c_2) = & \alpha_1 sim_{ts}(c_1.score, c_2.score) + \\ & \alpha_2 sim_{ts}(c_1.holes, c_2.holes) + \\ & \alpha_3 sim_{ts}(c_1.height, c_2.height) \end{aligned}$$

where the weights $\alpha_1, \alpha_2, \alpha_3$ can be adjusted to give more or less importance to each time series. The most important one in Tetris to predict the player skill level is the game score ($\alpha_1 = 0.70$), followed by the number of holes ($\alpha_2 = 0.25$) and finally the board height ($\alpha_3 = 0.05$). These optimal weights were calculated using grid search with increments of 0.05 and 10-fold cross validation to improve the accuracy of the classifier.

We compute the similarity between 2 time series using a similarity based on the Euclidean distance [28]:

$$sim_{ts}(r, s) = 1 - \sqrt{\sum_{i=1}^n (r_i - s_i)^2}$$

where r and s are time series of size n .

Finally, inconsistent player behavior during the game can produce frequent changes in the profile prediction, producing continuous changes in the difficulty of the game. For example, an expert player may be distracted at some point in the game and appear to be a beginner, or a novice player may have a good run for a while. Sudden changes in game difficulty are often perceived as "something strange is going on" and provoke a distrustful attitude in players. In addition, the profile predictions of this module can fail (the predictions become more accurate as the game progresses), so it is advisable to use some *inertia function* that avoids transitioning between extreme profiles. In our implementation, we use a inertia function that only allows transitions between adjacent profiles. That is, if a player was classified as a beginner in the previous performance window, she can only be classified as a beginner or intermediate in the current performance window, no matter how well she places the last pieces.

C. Difficulty Adjustment

The purpose of this component is to decide *when* and *how* to modify the difficulty of the game. We focus on 2 of the 9 dimensions that represent the optimal psychological state of flow: challenge-skill balance and sense of control. This way, the decision of whether to modify the difficulty of the game and with what intensity to do so depends on the player's profile and whether she is in control within her profile.

1) *Is the player in control?*: The sense of *control* indicates whether an individual feels capable of performing the task because it is at an appropriate difficulty level and has been identified as one of the most important components of the flow experience [11]. If the task is too difficult, the subject experiences anxiety, or if it is too easy, the subject experiences boredom, so the feeling of control is somewhere between those two extremes. To indirectly measure the control of the player, we compute the complexity of the game state and then compare it with the mean complexities of the *good*, *too easy*

and *too hard* games in our collection of labelled games from players within the same profile than the current player. For the player to be in control, that is, performing a task with the appropriate difficulty level, the complexity of the board during the last performance window should be more similar to the mean complexity of the good games than to the too easy or too hard ones. As in the previous component, we only compare the complexity of performance windows corresponding to the same moment (number of piece) in different games.

We compute the *game state complexity* after each piece is settled in its final location with the following formula:

$$C = \bar{a}^2 + \sigma(\text{contour}) + h$$

where $\bar{a}$ is the median height of the columns on the board squared, $\sigma(\text{contour})$ is the standard deviation of the height of the columns, and h is the number of holes between the pieces. These variables represent challenges the player must overcome as the speed of the falling piece increases: low board height, homogeneous board contour and few holes between the pieces.



Fig. 4. Mean game state complexity and its standard deviation for beginner players as the game progresses.

Figure 4 shows the mean game state complexity and its standard deviation for players with a beginner profile as the game progresses. The red, green and blue lines represent the mean complexity for the *too hard*, *good* and *too easy* games respectively. As the game progresses, the complexities of the games come apart and each one of them increases with a different speed. This different trajectories also exists within the average players but the lines are not so far apart.

Finally, to decide if the player is in control or not (a binary variable), we compare the complexity of each game state in the last performance window with the mean complexity of the *good*, *too easy* and *too hard* games in the same instant. The player will be in control when most of those complexities are closer to the good games than to the too easy or too hard games. If the player is not in control, the difficulty of the game is not at the appropriate level for her current skill level.

2) *Modify the difficulty in Tetris*: The difficulty in Tetris depends basically on two variables: the type of pieces delivered and the speed at which it falls. Using these two variables,

we can make the game more or less difficult and adapt it to the skill level of the current player.

The next piece is probably the most important event in the Tetris game. A good piece that fits the current configuration of the board can potentially reduce the board height, while a bad piece could greatly increment the complexity of the board. We assess how good each piece is for the current board with a heuristic function that evaluates each game board that can be reached by placing that piece in each possible final location. The heuristic value associated with each game board is the board height plus the number of holes, so the smaller the value the better the board. The heuristic value of each piece is the best (lowest) value of the reachable boards (so we assume the player will place it optimally). It is relevant to mention that it might not be a good idea to always provide the best or worst next piece, because some of them might be selected too frequently and be perceived negatively by the players. It is better to select one of the best or worst pieces randomly to ensure enough variability.

The difficulty of the game can also be changed by altering the speed at which the pieces fall. But we must do this carefully, because the falling speed is supposed to always increase in Tetris, and we don't want players to detect that we are changing the game. Also, if we reduce the drop speed too much, the game might never end. Therefore, the correct approach in this case is to slightly increase or decrease the drop speed increment each time the player places another piece.

Although our methodology can be used to make the game more or less difficult, in the experiment described next we have focused solely on making the game easier and thereby helping less experienced players. We think that beginner players have more trouble enjoying the game and getting into flow, while expert players already have the necessary skills to achieve the proposed challenges and need little or no help to have a satisfying experience. Therefore, we define two different adaptation's levels to produce a challenge-skill balance that consider both the player profile and the feeling of control:

- *Speed-piece adaptation* This level modifies the falling speed and the next piece. The next piece is selected randomly among the best 3 candidates. The falling speed rate increases by half of what it normally would. This adaptation level is active when the player is a beginner and is not in control.
- *Piece-only adaptation* This level modifies only the next piece. This adaptation level is active when either the player is a beginner and is in control, or when the player is average and is not in control.

IV. EXPERIMENT

In the experiment we compare three different versions of Tetris: normal, trivial and balanced (DDA). The *normal* version corresponds to the original Tetris Analytics system in which the next piece is chosen randomly, while the other two versions are modifications of this same system. The *normal* version is the original Tetris game without any modifications. The *trivial* version always returns a good next piece so that the player can easily make lines. The *balanced* version analyses

the player's performance and changes the difficulty level accordingly as we have explained in the previous section. The goal of this experiment is to measure the flow that players experience during the games.

The experiment will test the following hypotheses:

- H1: We get more players to experience flow using DDA (balanced version) than without it (normal version).
- H2: It is harder for beginners to experience flow compared to intermediate or advanced level players (normal version).
- H3: If we provide an easy Tetris version to players with a low skill level, then they will experience flow (trivial version).
- H4: If Tetris is too easy, then the players will not experience flow (trivial version).

A. Experimental setup

The experiment was performed on site with the presence of a moderator that provided the initial instructions and guided the experiment. 40 volunteers participated in the experiment (computer science students other than those who participated in Section III-A1). They were asked to fill out a form with basic demographic information (*age, sex*) and answer some questions regarding their use of video games: *Are you a regular player?* (yes/no), *How many hours do you play per week?* (0-5, 5-10, 10-15, 15-20, +20), *Do you know Tetris?* (yes/no), *What skill level do you have in Tetris?* (never played before, beginner, intermediate, advanced).

Each participant began by playing Tetris for 40 seconds to become familiar with the controls and to remember how the game works. Next, each participant played 3 complete games, filling out the SHORT Flow State Scale (S-FSS) [12] questionnaire after each game. Finally, each participant chose the game she liked the most (1, 2 or 3). The experiment was designed to last approximately 30 minutes.

The participants did not know the purpose of the experiment or whether the games corresponded to different versions of Tetris. Participants were randomly divided in 3 groups so that they played the different versions in different order:

- Group 1: trivial, balanced, normal.
- Group 2: normal, trivial, balanced.
- Group 3: balanced, normal, trivial.

The SHORT Flow State Scale (S-FSS) [12] determines whether a player has entered flow during an activity (Table I). It contains nine items, and the scores of each one represents a dimension of flow. The scale is designed as a post-event assessment of flow, with instructions written to connect the subject to a recently completed activity. A more accurate evaluation of the flow state is possible when the scale is administered close to the end of the game. Responses range from 1 (strongly disagree) to 5 (strongly agree). A low response value indicates that the subject's experience was not substantially of a flow nature. Conversely, a high response value symbolizes that the individual experienced flow substantially. The average score of 3 on the status scales represents a choice of "neither agree nor disagree". This average score may indicate some degree of approval, but it may also mean some ambiguity regarding the

item's relevance to the person's experience. Nevertheless, it is reasonable to interpret moderate level scores as neither strong evidence that the person has experienced flow nor strong proof that the person's experience did not include the flow.

B. Results

The age of the 40 participants ranged from 18 to 42 years, with a mean age of 27.3 years. More than half of the participants played video games more than 5 hours per week (see Figure 5). Also, more than half of the participants considered themselves beginners (see Figure 6).

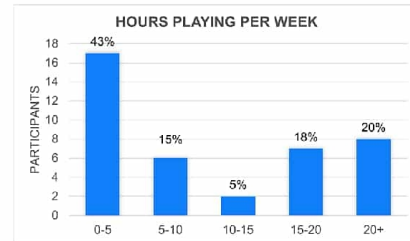


Fig. 5. Hours per week that participants play video games.

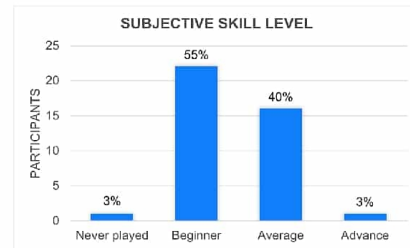


Fig. 6. Subjective skill level of the participants.

Let's start the analysis with the participants' favorite version of Tetris. In the last question of the experiment, participants chose which of the 3 games they liked the most. Figure 7 shows the percentage of times each version was chosen: 20% of the players selected the normal version, 27.5% selected the trivial version, and 52.5% selected the balanced version. If we analyze the results filtered by the player's subjective skill level of Tetris (the level they believed they had in the initial questionnaire), the balanced version was chosen twice as often as the next version (trivial) by both beginners and intermediate players. The advanced player also chose the balanced version, and the player who had never played Tetris before chose the trivial version.

The main goal of the experiment was to determine which version of Tetris generates the highest flow state. Since there were 40 participants and each of them played the 3 versions of Tetris, we had 40 SHORT Flow State Scale questionnaires for each version of Tetris. We compare Tetris versions in pairs in order to determine whether a version of Tetris produce a

TABLE I
EACH ITEM FROM THE SHORT FLOW STATE SCALE AND THE DIMENSION IT REPRESENTS [12].

#	Dimension	Sentence
1	Challenge-Skill Balance	I felt I was competent enough to meet the demands of the situation
2	Merging of Action and Awareness	I did things spontaneously and automatically without having to think
3	Clear Goals	I had a strong sense of what I wanted to do
4	Unambiguous Feedback	I had a good idea about how well I was doing while I was involved in the task/activity
5	Concentration on the Task at Hand	I was completely focused on the task at hand
6	Sense of Control	I had a feeling of total control over what I was doing
7	Loss of Self-Consciousness	I was not worried about what others may have been thinking of me
8	Transformation of Time	The way time passed seemed to be different from normal
9	Autotelic Experience	I found the experience extremely rewarding

TABLE II
AVERAGE FLOW STATE PER TETRIS VERSION AND P-VALUE.

	Average Flow			p-value		
	normal	balanced	trivial	normal vs balanced	balanced vs trivial	normal vs trivial
Flow score	3.778	4.000	4.003	0.001	0.973	0.033

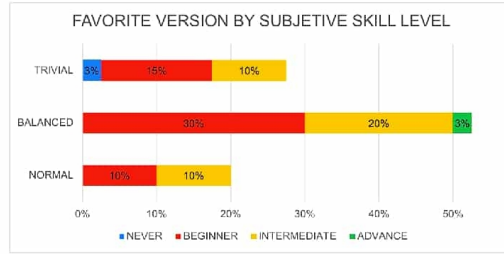


Fig. 7. Favorite version filtered by subjective skill level.

statistically significant higher flow state than other, by running a paired t-test for every considered pair of versions: normal vs. balanced; balanced vs. trivial; normal vs. trivial. Since we are running 3 different comparisons for the same dataset, we apply Bonferroni correction [29] and instead of considering a Type I error rate of 5% [30] we use a significance threshold of 0.017% ($\approx 0.05/3$).

As shown in Table II, the average flow score of the balanced version ($M=4$, $SE=0.602$) is significantly higher than the average flow score of the normal version ($M=3.778$, $SE=0.54$) with $p < .017$. Therefore, we accept *H1: we get more players to experience flow using DDA*.

To test *H2: it is harder for beginners to experience flow compared to intermediate or advanced level players*, we grouped participants according to their actual skill level (in contrast to the subjective skill level used before) using their final scores in the normal Tetris games. 21 participants were classified as beginners and 19 as intermediate players. We found no significant differences between the average flow scores in both groups of players, so we reject *H2*. However, if we separate the data and run again the paired t-test for beginners (Table III) and intermediate players (Table IV) separately, we get additional insights. For beginners, both the balanced and trivial versions provide significantly ($p < .017$) higher flow than the normal version, while for the intermediate

players it is only the case with the balanced version.

To test *H3: If we provide an easy Tetris version to players with a low skill level, then they will experience flow* we compared the trivial and normal versions of Tetris within the group of beginner players. Both the balanced and trivial versions of Tetris are easier than normal Tetris. The balanced version adapts the challenges to make them easier when necessary, while the trivial version always provides help. The average flow score of the trivial version ($M=4.037$, $SE=0.628$) is significantly higher than the average flow score of the normal version ($M=3.778$, $SE=0.415$) with $p < .017$. These results show that both versions providing easier challenges helped beginners to experience flow more strongly than in normal Tetris. Therefore, we accept *H3*.

Finally, we accept *H4: if Tetris is too trivial, then the players will not experience flow*, because the resulting flow in the trivial version is not significantly higher ($p < .017$) than the flow measured with the normal or the balanced versions in our experiment (see Table II). Nevertheless, if we consider just beginners (Table III) then flow in the trivial version is significantly higher than in the normal one, so we could reject *H4* for beginners.

C. Discussion

According to our results in this experiment, we can conclude that:

- The average flow experience when playing a balanced version is significantly higher than when playing the normal version.
- The average flow experience from the balanced version is not significantly higher than the trivial version.
- For beginners, both the trivial and balanced versions improve the flow experience compared to the normal version.
- We have no evidence that it is harder for beginners to get into flow compared to intermediate and advanced players.

We expected players to find the trivial version boring compared to the balanced version, but that was not the case. This is probably due to the fact that most of the participants

TABLE III
BEGINNERS' AVERAGE FLOW SCORE PER VERSION AND P-VALUE.

	Average Flow			Beginners		
	normal	balanced	trivial	normal vs balanced	p-value balanced vs trivial	normal vs trivial
Flow score	3.788	3.979	4.037	0.000	0.295	0.004

TABLE IV
INTERMEDIATE'S AVERAGE FLOW SCORE PER VERSION AND P-VALUE.

	Average Flow			Intermediate Players		
	normal	balanced	trivial	normal vs balanced	p-value balanced vs trivial	normal vs trivial
Flow score	3.766	4.023	3.965	0.003	0.689	0.159

in the experiment consider themselves beginners and had not played Tetris for a very long time, so they were happy to play an easy version of the game. Besides, participants scored better on the trivial version (average score 4720) than on the balanced one (average score 3659) and that can influence the answers to the questionnaire. In addition, each game lasts about 3-5 minutes and that is probably not long enough to get bored even if the game is very easy. We believe players would find the trivial version tedious if they had to play it more than a few minutes. Furthermore, the balanced version was chosen as the favorite one by 53% of the participants so we can conclude that the balanced Tetris version provides a more rewarding experience than the trivial one.

Regarding the limitations of the study, we have identified some aspects that should be further investigated in the future. First, the 40 participants were related to the computer field, so they may not be representative of the general population of gamers. Second, our study does not consider personal characteristics that cause some people to experience flow with greater or lesser difficulty and intensity. Third, our study focuses on beginner or intermediate players, and the balanced version that we use in the experiment only decreases the difficulty of the game, never increases it. It would remain to be studied how the dynamic difficulty adjustment affects expert players who need greater challenges. Finally, our experiment only considers very short game sessions. It would be interesting to study longer periods of time to see whether the difficulty of the game evolves in line with the player's improvement.

V. CONCLUSIONS AND FUTURE WORK

This work presents a novel methodology that serves as a guide to implement DDA in video games using ideas from the theory of flow. Our methodology is divided into 2 phases: pre-analysis and online game play. During pre-analysis we collect and analyze game traces to identify the most relevant game variables to assess the performance of the player. We also create a case base of game fragments labeled with the player's skill level and a satisfaction scores. The second phase takes place during the game and continuously predicts the current player's profile and decides when and how to adapt the difficulty of the game. To predict the player's profile we compare her performance during the last performance window with those stored in the case base. In turn, the decision whether

to alter the difficulty of the game depends on if the player is in control, that we measure indirectly by comparing the complexity of the last game states with the average complexity of the games in the case base that were rated as good, too easy or too difficult by other players with the same profile.

We have implemented this methodology in Tetris and performed an experiment that shows that the average flow experience is significantly higher using DDA than in the normal version. We found no overall significant difference between the flow experienced between the DDA and trivial versions. We think this is because the participants were mostly beginners with little experience playing Tetris and only played one trivial game that lasted just 3 or 4 minutes and that was not enough to find it tedious. Nevertheless, differences between trivial and balanced versions can be found if we study beginners and intermediate players separately. While we found a significant difference for both versions with respect to the normal one in beginners, for intermediate players we only found the balanced version to be better than the normal one.

Although this work focuses on a particular game, we believe that the proposed methodology is general enough to be applied to a wide range of games. The main decisions that must be made to use this methodology in a particular game are (1) to identify the most relevant game variables to assess the performance of the player during the game, (2) to decide the length of the performance windows used for DDA, (3) to define a function to measure the complexity of the game during a performance window, and (4) to decide how to make the game more difficult or easier depending on the player's needs at any given time. For example, in a classic game such as Space Invaders the player's performance could be measured using variables such as the game score, the number of ships and aliens destroyed, the number of shots that did not hit any enemy, ...; new performance windows could be created every 10, 20 or 30 seconds depending on how often we want to reevaluate the difficulty of the game, with values collected every few seconds; the complexity of the game could be measure using the speed of the aliens, their distance to the ground, the amount of barriers remaining for the player to cover, ...; and the difficulty of the game could be changed by altering the speed of the aliens or their fire rate. Of course, these are just ideas that should be tested and evaluated by both game designers, who know in detail how the different game

mechanics work, and data analysts who can find and verify patterns in the game traces.

Our proposal has some limitations as well. We use a case-based implementation and, therefore, we require a good case base that effectively represents different types of players and games (with a good coverage and diversity of cases). In addition, the decision of using ex-post questionnaires instead of real-time sensors to measure the player experience has some advantages (it is a not intrusive approach and it does not alter the player's experience), but it also means that we only get feedback about the whole game (or at least a level of the game) so we cannot differentiate which specific parts induced more or less flow in the player. As part of the future work, we would like to research the use of non-intrusive devices to identify and label good, too easy and too hard game fragments in the case base. Note that those devices would only be needed during the construction of the case base and not to perform DDA once the game is released. The spectacular progress of smart watches and bracelets in recent years makes us feel hopeful in this regard. In fact, the use of these devices could also help us to measure the effectiveness of a difficulty adjustment by examining the subsequent biometric response of the player.

Although the methodology presented in this paper is general, we have only tested it with beginner and intermediate players making the game easier and never more difficult. We would also like to test this type of DDA during longer periods of time to see if the difficulty of the game evolves in line with the player's improvement. We are interested in the use of DDA in serious games where keeping the player engaged is critical to enhance learning over longer periods of time.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Committee of Science and Innovation (PID2020-114596RB-C21) and the Complutense University of Madrid (Group 921330). We would like to thank the anonymous reviewers for their helpful and insightful comments.

REFERENCES

- [1] A. Denisova, P. A. Cairns, C. Guckelsberger, and D. Zendle, "Measuring perceived challenge in digital games: Development & validation of the challenge originating from recent gameplay interaction scale (COR-GIS)," *Int. J. Hum. Comput. Stud.*, vol. 137, p. 102383, 2020.
- [2] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller, "User-centered design in games," in *The human-computer interaction handbook*. CRC Press, 2002, pp. 915–938.
- [3] D. Charles, A. Kerr, and M. McNeill, "Player-centred game design: Player modelling and adaptive digital games," in *Proceedings of the Digital Games Research Conference*, vol. 285, 2005, pp. 285–298.
- [4] R. Hunnicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM, 2005, pp. 429–433.
- [5] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *Discovery Science, 12th International Conference, DS 2009, Porto, Portugal, October 3-5, 2009*, 2009, pp. 197–211.
- [6] A. Ram, S. Ontañón, and M. Mehta, "Artificial intelligence for adaptive computer games," in *FLAIRS Conference*, 2007, pp. 22–29.
- [7] R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations: a survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 85–99, 2011.
- [8] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.
- [9] M. Csikszentmihalyi, *Flow: The psychology of optimal experience*. Harper & Row New York, 1990.
- [10] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Commun.*, vol. 7, no. 1, pp. 39–59, 1994.
- [11] J. Nakamura and M. Csikszentmihalyi, "The concept of flow," in *Flow and the foundations of positive psychology*. Springer, 2014, pp. 239–263.
- [12] S. Jackson, B. Eklund, and A. Martin, "The flow manual the manual for the flow scales manual. sampler set," *Mind*, vol. 2011, pp. 1–85, 2011.
- [13] C. A. Cruz and J. A. R. Uresti, "Player-centered game ai from a flow perspective: Towards a better understanding of past trends and future directions," *Entertainment Computing*, vol. 20, pp. 11–24, 2017.
- [14] P. Sweetser, D. M. Johnson, P. Wyeth, A. Anwar, Y. Meng, and A. Ozdowska, "Gameflow in different game genres and platforms," *Comput. Entertain.*, vol. 15, no. 3, pp. 1:1–1:24, 2017.
- [15] G. Chaneil, C. Rebetez, M. Bétrancourt, and T. Pun, "Emotion assessment from physiological signals for adaptation of game difficulty," *IEEE Trans. Syst. Man Cybern. Part A*, vol. 41, no. 6, pp. 1052–1063, 2011.
- [16] M. Klasen, R. Weber, T. T. Kircher, K. A. Mathiak, and K. Mathiak, "Neural contributions to flow experience during video game playing," *Social cognitive and affective neuroscience*, vol. 7, no. 4, pp. 485–495, 2012.
- [17] S. Park, H. Sim, and W. Lee, "Dynamic game difficulty control by using eeg-based emotion recognition," *International Journal of Control and Automation*, vol. 7, pp. 267–272, 03 2014.
- [18] A. Stein, Y. Yotam, R. Puzis, G. Shani, and M. Taieb-Maimon, "Eeg-triggered dynamic difficulty adjustment for multiplayer games," *Entertain. Comput.*, vol. 25, pp. 14–25, 2018.
- [19] T. Alves, S. Gama, and F. S. Melo, "Flow adaptation in serious games for health," in *6th IEEE International Conference on Serious Games and Applications for Health, SeGAH 2018, Vienna, Austria, May 16-18, 2018*, J. L. Vilaça, T. Grechenig, D. Duque, N. F. Rodrigues, and N. Dias, Eds. IEEE, 2018, pp. 1–8.
- [20] S. A. Jackson, A. J. Martin, and R. C. Eklund, "Long and short measures of flow: The construct validity of the fss-2, dfs-2, and new brief counterparts," *Journal of Sport and Exercise Psychology*, vol. 30, no. 5, pp. 561–587, 2008.
- [21] K. Spiel, S. Bertel, and F. Kayali, "'not another Z piece!': Adaptive difficulty in TETRIS," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*, G. Mark, S. R. Fussell, C. Lampe, m. c. schraefel, J. P. Hourcade, C. Appert, and D. Wigdor, Eds. ACM, 2017, pp. 5126–5131.
- [22] A. Nagle, P. Wolf, and R. Riener, "Towards a system of customized video game mechanics based on player personality: Relating the big five personality traits with difficulty adaptation in a first-person shooter game," *Entertain. Comput.*, vol. 13, pp. 10–24, 2016.
- [23] M. P. Silva, V. do Nascimento Silva, and L. Chaimowicz, "Dynamic difficulty adjustment on MOBA games," *Entertain. Comput.*, vol. 18, pp. 103–123, 2017.
- [24] J. E. Cechanowicz, C. Gutwin, S. Bateman, R. L. Mandryk, and I. Stavness, "Improving player balancing in racing games," in *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play, Toronto, ON, Canada, October 19 - 22, 2014*, L. E. Nacke and T. C. N. Graham, Eds. ACM, 2014, pp. 47–56.
- [25] O. Missura and T. Gärtner, "Predicting dynamic difficulty," in *Advances in Neural Information Processing Systems*, 2011, pp. 2007–2015.
- [26] A. Zook and M. O. Riedl, "A temporal data-driven player model for dynamic difficulty adjustment," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, USA, October 8-12, 2012*, M. Riedl and G. Sukthankar, Eds. The AAAI Press, 2012.
- [27] D. S. L. Ariza, A. A. Sánchez-Ruiz, and P. A. González-Calero, "Time series and case-based reasoning for an intelligent tetris game," in *Case-Based Reasoning Research and Development - 25th International Conference, ICCBR 2017, Trondheim, Norway, June 26-28, 2017, Proceedings*, ser. Lecture Notes in Computer Science, D. W. Aha and J. Lieber, Eds., vol. 10339. Springer, 2017, pp. 185–199.
- [28] S. Rani and G. Sikka, "Recent Techniques of Clustering of Time Series Data: A Survey," *International Journal of Computer Applications*, vol. 52, no. 15, pp. 1–9, 2012.
- [29] N. J. Salkind, Ed., *Encyclopedia of research design*. SAGE Publications, Inc., 2010.
- [30] J. de Winter and D. Dodou, "Five-point likert items: t test versus mann-whitney-wilcoxon," *Practical Assessment, Research and Evaluation*, vol. 15, pp. 1–16, 2010.