# 俄罗斯方块游戏项目开发文档

2025.12.2

**23542854 胡永杰**

# 一、项目名称

**Classic Tetris Game（经典俄罗斯方块游戏）**

基于DOSBox+MASM环境开发的16位汇编语言俄罗斯方块游戏

---

# 二、项目功能介绍

## 2.1 核心游戏功能

### 2.1.1 方块系统

- **7种方块类型**：正方形、四格横条、L形、T形、Z形（两种旋转状态）
- **随机生成**：使用系统时钟实现伪随机数生成，确保方块类型随机出现
- **颜色系统**：方块下落时切换不同颜色显示，增强视觉体验

### 2.1.2 方块操作

- **左右移动**：使用方向键或A/D键控制方块水平移动
- **快速下落**：使用方向键下键加速方块下落
- **旋转功能**：使用方向键上键实现方块旋转
- **碰撞检测**：完整的边界检测和方块重叠检测系统

### 2.1.3 游戏逻辑

- **自动下落**：方块自动向下移动
- **行消除**：当一行完全填满时自动消除该行，上方方块下移
- **游戏结束判定**：当方块堆积到顶部无法放置新方块时游戏结束
- **连续游戏**：方块落底后自动生成新方块

## 2.2 游戏模式

### 2.2.1 普通模式（Normal Mode）

- 固定下落速度
- 难度级别始终为0
- 适合新手熟悉游戏操作

### 2.2.2 难度模式（Difficulty Mode）

- **动态难度调整**：每20秒自动提升一个难度级别
- **速度递增**：随着难度级别增加，方块下落速度逐渐加快
- **难度显示**：实时显示当前难度级别（Level 1, 2, 3...）

- **最好成绩记录**：自动记录并显示难度模式下的最高分数

## 2.3 界面系统

### 2.3.1 开始菜单

- **标题**："TETRIS"艺术字标题
- **模式选择**：
  - [1] Normal Mode（普通模式）
  - [2] Difficulty Mode（难度模式）
  - [3] Exit（退出游戏）
- **最好成绩显示**：显示难度模式下的历史最好成绩
- **彩色界面**：使用亮黄色边框、亮绿色文字、亮紫色选项标记

### 2.3.2 游戏界面

- **左侧游戏区域**：40x18的游戏主区域，带有边框
- **右侧信息面板**：
  - Level（难度级别）：显示当前难度
  - Time（游戏时间）：显示MM:SS格式的游戏时长
  - Score（分数）：显示当前得分，消除一行得10分
  - Operation（操作说明）：显示按键操作提示
  - System（系统说明）：显示开始/退出操作

### 2.3.3 游戏结束界面

- **"GAME OVER"**：醒目的游戏结束提示
- **按任意键返回**：游戏结束后可按任意键返回开始菜单

## 2.4 计分与计时系统

### 2.4.1 分数系统

- **计分规则**：每消除一行获得10分
- **多位数显示**：支持显示0-65535的分数
- **分数显示**：实时更新分数显示

### 2.4.2 计时系统

- **游戏计时**：从游戏开始时刻开始计时，而非显示系统时间
- **时间格式**：MM:SS格式（分钟:秒）
- **精度**：基于BIOS时钟中断（约18.2次/秒）

### 2.4.3 最好成绩系统

- **自动记录**：难度模式下自动记录最高分数
- **持久化显示**：在开始菜单显示历史最好成绩
- **实时更新**：当打破记录时自动更新

# 三、项目开发流程及心得

## 3.1 开发环境

- **开发工具**：MASM (Microsoft Macro Assembler)
- **运行环境**：DOSBox（DOS模拟器）
- **编程语言**：16位x86汇编语言
- **内存模型**：`.model small`（小内存模型）
- **显示模式**：80x25文本模式，使用BIOS中断操作显存

## 3.2 开发流程

**阶段一：基础框架搭建**

**目标**：实现基本的游戏窗口和方块显示

**主要工作**：

- 设计游戏界面布局（边框、信息面板）
- 实现清屏和界面绘制功能
- 实现方块数据结构和绘制函数
- 使用BIOS中断（int 10h）进行显示操作

**遇到问题**：

- 屏幕显示异常（黑屏、光标位置错误）
- 方块绘制位置计算错误
- 字符编码和颜色属性混淆

**解决方案**：

- 正确初始化段寄存器（ES指向显存0B800h）
- 使用段地址方式而非绝对地址方式显示界面
- 区分字符字节和颜色属性字节（每个字符占2字节）

**阶段二：游戏核心逻辑**

**目标**：实现方块下落、移动、旋转等核心功能

**主要工作**：

- 实现方块随机生成（`rand_block`函数）
- 实现方块下落逻辑（`move_block`函数）
- 实现碰撞检测（`judge_block`函数）
- 实现键盘输入处理（BIOS int 16h）
- 实现方块旋转（`block_change`函数）

**遇到问题**：

- 编译错误："jump out of range"（跳转超出范围）
- 操作数类型不匹配错误

- 方块移动速度过慢，无法快速移动
- 旋转功能失效

**解决方案**：

- 使用`jmp far ptr`或重构代码结构解决跳转范围问题
- 统一数据宽度（db vs dw），特别注意地址存储使用dw
- 优化输入处理：引入`HandleInputLoop`实现高频轮询
- 分离旋转逻辑：修正旋转键检测（仅使用方向键上）

**技术要点**：

- 使用`MOVE_TICKS`常量控制输入轮询频率
- 实现非阻塞键盘输入（`int 16h, AH=01h`检查，`AH=00h`读取）
- 方块数据结构：使用相对坐标数组定义方块形状

**阶段三：行消除与计分**

**目标**：实现完整的游戏逻辑，包括行消除和计分

**主要工作**：

- 实现满行检测（`get_score`函数）
- 实现行消除和方块下移（`clear_a`函数）
- 实现分数计算和显示（`add_score`、`show_score_display`函数）
- 修复分数显示（支持多位数显示）

**遇到问题**：

- 满行检测逻辑错误（检查空格而非方块字符）
- 行消除后上方方块不下移
- 分数显示不正确（只能显示个位数）
- 分数在方块落底后被重置

**解决方案**：

- 修正检测逻辑：检查`'@'`字符（方块显示字符）而非空格
- 修正消除逻辑：跳过边框字符（`*`, `|`, `+`, `-`），正确处理行移动
- 重构分数显示：使用除法取余和栈实现多位数转换
- 使用条件判断避免分数在游戏进行中被重置

**技术要点**：

- 分数变量类型改为`dw`（支持0-65535）
- 使用栈实现数字到ASCII的转换
- 清空旧显示区域避免数字重叠

**阶段四：难度系统与计时**

**目标**：实现难度模式和游戏计时功能

**主要工作**：

- 设计难度系统架构（普通模式vs难度模式）
- 实现游戏计时（使用BIOS int 1Ah获取系统时钟）
- 实现难度级别动态调整（每20秒升级）
- 实现下落速度动态调整（根据难度级别）
- 实现最好成绩记录

**遇到问题**：

- 时间显示重置（每次方块落底后时间清零）
- 难度级别继承（新游戏继承上一局的难度）
- 最好成绩不记录（主动退出时未更新）
- 难度模式分数上限（超过10000后不增加）

**解决方案**：

- 使用全局变量`game_start_time`记录游戏开始时间，仅在首次调用时初始化
- 在模式选择时重置所有难度相关变量（`game_start_time`、`last_level_time`）
- 创建`update_best_score`函数，在退出时统一检查更新
- 移除分数显示位数限制，支持大数字显示

**技术要点**：

- 32位时间计算（使用CX:DX存储时钟滴答数）
- 时间转换：滴答数→秒数→MM:SS格式
- 难度速度公式：`delay = 3000 - level * 150`（最小500）
- 使用条件判断区分普通模式和难度模式

**阶段五：界面优化与代码清理**

**目标**：优化界面显示，清理冗余代码

**主要工作**：

- 重新设计开始菜单和结束界面（更美观的ASCII艺术）
- 优化游戏界面边框样式
- 调整数据显示位置对齐（Level、Time、Score对齐）
- 删除未使用的函数和变量（`read_score`、`write_score`等）
- 代码注释和格式优化

**遇到问题**：

- 数据显示不对齐（Level、Time、Score列位置不一致）
- 存在未使用的代码段（冗余函数和变量）

**解决方案**：

- 统一所有数据显示的起始列位置（第56列）
- 全面检查代码，删除所有未使用的函数和数据定义

---

## 3.3 核心技术实现

#### 3.3.1 显示系统

- **显存操作**：直接操作0B800h段地址的显存
- **字符属性**：每个字符占2字节（字符码+颜色属性）
- **颜色编码**：使用8位颜色属性（前景色+背景色+闪烁位）

#### 3.3.2 输入处理

- **非阻塞输入**：使用`int 16h, AH=01h`检查按键
- **连续输入**：使用循环轮询实现平滑移动
- **扫描码处理**：区分ASCII码和扫描码（方向键使用扫描码）

#### 3.3.3 游戏循环

- **主循环**：`start_game` → `move_block` → `HandleInputLoop` → `start_game`
- **定时控制**：使用循环延时和BIOS时钟中断控制游戏速度
- **状态管理**：使用全局变量管理游戏状态（位置、类型、分数等）

#### 3.3.4 碰撞检测

- **边界检测**：检查方块是否超出游戏区域边界
- **重叠检测**：检查方块是否与已放置方块重叠
- **实时验证**：每次移动和旋转前进行碰撞检测

---

### 3.4 开发心得

#### 3.4.1 汇编语言学习感悟

**从理论到实践的跨越**：

- 本学期通过课堂学习掌握了汇编语言的基本语法、指令集和程序结构
- 然而，理论知识只有通过实际项目才能真正理解和应用
- 这个俄罗斯方块项目让我将课堂上学到的寄存器操作、内存管理、中断调用等知识点串联起来，形成了完整的知识体系
- 真正理解了"纸上得来终觉浅，绝知此事要躬行"的道理

**汇编语言的独特魅力**：

- 汇编语言虽然编写复杂，但能够直接控制硬件，对计算机底层有最直观的理解
- 每一条指令都有明确的目的，能够清楚地看到程序是如何在CPU和内存之间工作的
- 这种接近硬件的编程方式，让我对计算机系统有了更深刻的认识
- 相比高级语言，汇编语言需要自己管理内存、寄存器，培养了更严谨的编程思维

**期末项目的意义**：

- 作为本学期的汇编语言课程期末项目，这个游戏项目不仅是知识的综合运用，更是学习成果的集中体现
- 项目涵盖了本学期课程的核心内容：指令系统、程序结构、中断调用、内存管理、I/O操作等
- 通过完成这个项目，不仅巩固了课堂知识，还拓展了学习范围，如游戏循环设计、状态管理等实用技能
- 项目开发过程中遇到的问题和解决方法，都是宝贵的实践经验

### 3.4.2 汇编语言特性理解

**段寄存器的重要性**：

- 深刻理解了16位汇编中段寄存器（CS、DS、ES、SS）的作用
- CS（代码段）指向程序代码，DS（数据段）指向程序数据，ES（附加段）常用于字符串操作和显存操作，SS（栈段）指向栈空间
- 显存操作必须正确设置ES寄存器指向0B800h，这是文本模式显存的固定地址
- 数据访问必须确保DS指向正确的数据段，否则会出现内存访问错误
- 在函数调用时，需要正确保存和恢复段寄存器，否则会导致程序崩溃

**内存模型**：

- `.model small`适合大多数DOS程序，代码段和数据段分别限制在64KB内
- 必须明确定义数据段、代码段、栈段，这是汇编程序的基本结构
- 使用`assume`伪指令告诉汇编器段的使用关系，帮助编译器进行类型检查
- 理解段：偏移地址的内存寻址方式，这是16位实模式的核心概念

**数据宽度的重要性**：

- `db`（字节，8位）和`dw`（字，16位）的区别至关重要，错误的数据宽度会导致数据丢失或内存溢出
- 地址必须使用`dw`存储（16位偏移地址），这是本项目中的一个关键教训
- 分数等需要较大数值范围的变量应使用`dw`，否则无法支持超过255的值
- 在进行算术运算时，必须注意数据宽度匹配，否则会出现截断错误

**指令的理解**：

- 每条指令都有明确的含义和用途，不能随意使用
- 理解了MOV、ADD、SUB、CMP、JMP等基本指令的工作原理
- 掌握了条件跳转指令（JE、JNE、JA、JB等）的使用场景
- 学会了使用LOOP、CALL、RET等控制流指令实现循环和函数调用

### 3.4.3 BIOS中断的使用

**显示中断（int 10h）**：

- `AH=00h`：设置显示模式，本项目使用80x25文本模式（模式3）
- `AH=02h`：设置光标位置，用于在指定位置显示字符
- `AH=09h`：在当前光标位置写字符（不移动光标），适合批量写入
- `AH=0Eh`：在当前光标位置写字符（移动光标），适合逐个字符输出
- 直接操作显存（0B800h段）比使用BIOS中断更高效，这是游戏开发中的常用技巧

**键盘中断（int 16h）**：

- `AH=00h`：阻塞读取按键（等待按键），适合需要用户输入的场景
- `AH=01h`：非阻塞检查按键（不等待），适合游戏中的实时按键检测
- 理解扫描码和ASCII码的区别，方向键等特殊键只有扫描码
- 实现了非阻塞键盘输入，这是游戏能够流畅响应的关键

**时钟中断（int 1Ah）**：

- `AH=00h`：获取系统时钟滴答数（CX:DX），返回32位的时钟计数
- 滴答频率约18.2次/秒，需要除以18.2才能得到秒数
- 使用32位运算进行时间差计算，避免溢出问题
- 实现了游戏计时功能，这是难度系统的基础

#### 3.4.4 调试技巧与经验

**编译错误处理**：

- "jump out of range"：使用`jmp far ptr`或重构代码结构，将大函数拆分为小函数
- "operand types must match"：检查数据宽度是否匹配，特别注意db和dw的混用
- "symbol redefined"：注意MASM大小写不敏感，避免标签名冲突，使用更具体的命名
- "undefined symbol"：检查标号名称拼写，注意标号定义的可见性
- 学会了阅读MASM的错误信息，能够快速定位问题所在

**运行时错误调试**：

- 使用简单字符标记代码执行路径（如`mov es:[0], 'A'`），帮助追踪程序执行流程
- 检查寄存器值是否正确，使用DEBUG工具查看寄存器状态
- 注意栈平衡（push和pop必须匹配），栈不平衡会导致程序崩溃
- 使用断点调试，逐步执行代码，观察变量和寄存器的变化
- 学会了使用DEBUG的T（单步执行）、G（执行到断点）、D（查看内存）等命令

**逻辑错误调试**：

- 当程序运行结果不符合预期时，需要仔细分析算法逻辑
- 通过添加调试输出，观察关键变量的值，找出逻辑错误
- 使用注释标记可疑代码，逐一验证每个函数的正确性
- 学会了将复杂问题分解为简单问题，逐个解决

#### 3.4.5 游戏开发经验

**模块化设计**：

- 将功能拆分为独立过程（函数），每个函数负责单一功能
- 清晰的函数命名和注释，便于理解和维护
- 函数之间的接口设计要合理，尽量减少参数传递
- 学会了自顶向下的设计方法，先设计整体框架，再实现具体功能

**状态管理**：

- 使用全局变量管理游戏状态（方块位置、类型、分数、难度等）
- 状态初始化必须在合适的时机进行，避免状态混乱
- 注意状态重置时机（新游戏开始时），确保每次游戏都是干净的开始
- 理解了状态机在游戏开发中的重要性

**性能优化**：

- 减少不必要的循环和计算，特别是在游戏主循环中
- 直接操作显存比BIOS中断更高效，这是游戏流畅运行的关键
- 使用位操作和查表优化颜色设置，提高绘制效率

- 理解了性能优化需要在正确性保证的前提下进行

**用户体验**：

- 界面美观性很重要（ASCII艺术、颜色搭配），影响游戏的第一印象
- 操作响应速度影响游戏体验（高频输入轮询），延迟会让玩家感到不流畅
- 信息反馈要清晰（分数、时间、难度显示），让玩家了解游戏状态
- 理解了用户体验在游戏开发中的重要性

#### 3.4.6 学习过程与方法

**循序渐进的学习方法**：

- 项目开发分为五个阶段，每个阶段都有明确的目标和功能
- 从简单的界面显示开始，逐步添加游戏逻辑，最后完善用户体验
- 避免了试图一次性实现所有功能的冲动，降低了开发难度
- 每个阶段完成后都有明确的可运行版本，增强了学习成就感

**问题驱动的学习**：

- 在开发过程中遇到各种问题，通过解决这些问题学到了很多知识
- 遇到编译错误时，学会了阅读错误信息，查找资料，理解错误原因
- 遇到逻辑错误时，学会了调试方法，分析问题，逐步定位错误
- 理解了"错误是最好的老师"这句话的含义

**知识整合与应用**：

- 将课堂上学到的零散知识点整合到实际项目中
- 学习了游戏开发的相关知识，如游戏循环、状态管理、碰撞检测等
- 理解了理论知识如何转化为实际应用
- 形成了完整的知识体系

**文档与代码管理**：

- 学会了编写详细的开发文档，记录开发过程和遇到的问题
- 学会了代码版本管理，保留了多个版本的备份
- 理解了代码注释的重要性，便于后续维护和理解
- 学会了总结和反思，提炼开发经验

#### 3.4.7 项目难点与突破

**最难实现的功能**：

1. **方块旋转系统**：需要正确处理所有方块类型的旋转状态，特别是Z形和L形。通过定义每个方块的两个旋转状态，使用查表法实现旋转，避免了复杂的坐标变换计算。

2. **行消除算法**：检测满行、消除行、下移上方方块的逻辑较为复杂。通过仔细分析显存布局，使用循环和条件判断实现满行检测，通过内存复制实现行下移。

3. **难度动态调整**：需要协调时间、难度级别、下落速度三者关系。通过使用全局变量记录游戏开始时间和当前难度，在计时函数中动态计算难度级别，实现了平滑的难度过渡。

4. **数据显示对齐**：Level、Time、Score需要对齐显示，计算字符位置需要仔细。通过统一使用第56列作为数据显示起始位置，实现了完美的对齐效果。

**最大的挑战**：

- **汇编语言的底层特性**：没有高级语言的数据结构和库函数支持，所有功能都需要从底层实现。这要求对计算机系统有深入的理解。
- **内存管理的复杂性**：需要手动管理内存，注意段寄存器的设置，避免内存访问错误。一个小的错误就可能导致程序崩溃。
- **调试的困难性**：汇编程序的调试比高级语言困难得多，需要理解每条指令的执行效果，分析寄存器和内存的状态。
- **知识面的广度**：不仅需要掌握汇编语言，还需要理解BIOS中断、显存结构、键盘扫描码、时钟系统等硬件知识。

**最有价值的收获**：

1. **深入理解计算机底层工作原理**：
    - 理解了CPU如何执行指令，寄存器如何工作
    - 理解了内存的组织方式和访问方法
    - 理解了中断机制和I/O操作的本质
2. **掌握了BIOS中断的使用方法**：
    - 学会了使用显示中断进行屏幕输出
    - 学会了使用键盘中断进行输入处理
    - 学会了使用时钟中断进行时间管理
3. **培养了严谨的编程思维**：
    - 学会了仔细分析问题，设计解决方案
    - 学会了编写清晰的代码，添加必要的注释
    - 学会了调试程序，定位和修复错误
4. **体验了完整的游戏开发流程**：
    - 从需求分析到功能实现，从界面设计到用户体验优化
    - 理解了游戏开发中的各个环节和注意事项
    - 掌握了游戏循环、状态管理、碰撞检测等游戏开发核心技术
5. **提升了自主学习能力**：
    - 学会了查找资料，阅读文档，理解技术细节
    - 学会了分析问题，提出假设，验证解决方案
    - 学会了总结经验，提炼知识，形成自己的知识体系
6. **增强了解决问题的能力**：
    - 面对复杂问题时，学会了分解问题，逐个解决
    - 遇到错误时，学会了分析错误，查找原因，解决问题

    - 理解了"失败是成功之母"，从错误中学习和成长

#### 3.4.8 对汇编语言课程的整体反思

**课程学习的价值**：

- 汇编语言虽然在实际开发中使用较少，但它是理解计算机系统的关键
- 通过学习汇编语言，对高级语言的工作原理有了更深刻的理解
- 掌握了底层编程的基本技能，为学习操作系统、编译原理等课程打下了基础

**理论与实践的结合**：

- 课堂理论学习是基础，但只有通过实际项目才能真正掌握知识
- 本项目将课堂上学到的知识点串联起来，形成了完整的知识体系
- 理解了理论知识的实际应用场景和意义

**学习方法的重要性**：

- 循序渐进、问题驱动、知识整合、文档管理等学习方法都非常有效
- 学会了如何学习新技术，如何解决问题，如何总结经验
- 这些学习方法不仅适用于汇编语言，也适用于其他课程的学习

**对未来的展望**：

- 汇编语言的知识为后续学习操作系统、嵌入式系统、逆向工程等课程奠定了基础
- 严谨的编程思维和调试能力是程序员必备的基本素质
- 通过本项目获得的经验和技能将伴随整个编程生涯

---

### 3.5 项目开发反思

**做得好的地方**：

- 功能完整，包含了俄罗斯方块的所有核心功能
- 代码结构清晰，模块化设计合理
- 界面美观，用户体验良好
- 文档详细，记录了开发过程和遇到的问题

**可以改进的地方**：

- 代码注释可以更加详细
- 某些函数可以进一步优化性能
- 可以添加更多的游戏功能，如暂停、音效等
- 可以进一步优化界面显示效果

**经验总结**：

- 项目开发要有清晰的计划和阶段性目标
- 遇到问题要冷静分析，查找资料，寻求帮助
- 代码要注重可读性和可维护性
- 文档和注释同样重要，便于后续维护

---

# 四、技术规格

## 4.1 系统要求

- **操作系统**：DOS或DOSBox模拟器
- **内存要求**：至少640KB常规内存
- **显示模式**：80x25文本模式

## 4.2 开发工具

- **汇编器**：MASM 6.0或更高版本
- **链接器**：LINK（随MASM提供）
- **调试工具**：DEBUG或TD（Turbo Debugger）

## 4.3 编译方法

```
masm block.asm;
link block.obj;
block.exe
```

## 4.4 操作方法

- **开始游戏**：选择模式后按Enter
- **左右移动**：方向键左右或A/D键
- **快速下落**：方向键下键
- **旋转**：方向键上键
- **退出游戏**：Q键

---

# 五、项目总结

本项目成功实现了一个功能完整的俄罗斯方块游戏，不仅包含了经典俄罗斯方块的所有核心功能，还加入了难度系统、计时系统、最好成绩记录等增强功能。通过这个项目，深入学习了16位汇编语言编程、BIOS中断使用、显存操作、游戏循环设计等知识，提升了底层编程能力和系统理解能力。

虽然汇编语言开发效率较低，调试困难，但通过这个项目获得的底层系统知识和严谨的编程思维是无价的。这也为后续学习打下了坚实的基础。