俄罗斯方块

小组成员信息

陈道凡 生22 2022011312
谢宇涵 生22 2022011309

项目运行环境

所需最低c++标准库：c++17及以上
系统类型：windows
系统版本：windows11
编译器版本：Visual Studio 2022 (v143)

项目编译

直接使用vs提供的生成功能，需要注意的是，生成的.exe文件并不能直接运行，而是需要手动将资
源目录res/拷贝到可执行文件的同一目录下，这样才能实现图片资源的显示。

实现思路

使用easyx标准库，基于面向对象思想分别实现游戏主菜单、游戏界面、资源加载、文件配置等功能。
列表如下：

```
Begin_frame: 游戏主菜单，实现了新建和加载配置文件、新建和加载残局、开始游戏、退出游戏
Frame: 游戏界面，实现了随时暂停、残局生成、暂停后退出
Block: 游戏方块，实现了坐标控制、颜色显示
Animation: 加载游戏所需要的图片资源
EndGame: 专门处理新建残局和加载残局以及残局生成
```

游戏说明

主菜单操作说明：

```
("按 'G' 开始游戏");
("按 'Q' 退出游戏");
("按 'I' 新建配置");
("按 'U' 加载配置");
("按 'N' 创建残局");
("按 'M' 加载残局");
```

每一个按键按下后，会弹出控制台窗口，需要在控制台中输入文件名称来进行配置的创建和加载、残局的创建和加载。
注：
残局手动生成配备了残局生成窗口，可以在控制台中输入创建和删除方块的指令，实时显示当前地图方块的状态。

游戏界面操作说明：

```
（按‘ECS’暂停）
（按‘G’生成残局）
（按‘P’退出游戏）
```

暂停按键随时可以按下，只有在按下暂停后才能进行生成残局和退出游戏的操作，且暂停后仍可再次按下暂停键继续游戏。
注：
残局生成后可以选择继续进行游戏，暂停时的残局情况已经保存为文件；
按下G键后，需要在控制台中输入文件名

功能实现逻辑

基于面向对象的思想，将主菜单、游戏界面、资源、残局生成、方块封装为五个类。
主菜单负责配置文件和残局文件的创建和加载，并添加了开始游戏和退出游戏的消息处理。
游戏界面中接收方块的操作消息，暂停消息、生成残局消息和退出消息。
资源全部放在一起，在程序运行开始时初始化一次，防止重复加载资源，每个界面在初始化时获得资源指针，访问需要的资源。
残局内部具有生成残局、自定义残局、加载残局的功能，根据需要实现了不同的构造函数，每次被调用时生成特定残局。

`Animation`类

成员列表：

```
	IMAGE back_ground;
	std::vector<IMAGE*> block_png;
	std::vector<IMAGE*> block_group_png;
```

方法列表：

```
	/*
	* @biref 初始化所有的图片，仅初始化一次
	*/
	Animation();

	~Animation();

	/*
	* @brief 加载方块图片
	*/
	void load_block_png();

	/*
	* @brief 加载方块组合图片
	*/
	void load_block_group_png();
```

实现逻辑：
在资源类中，封装了方块图片、方块组合图片、背景图片等信息，对于需要用到资源的类，在初始化的时候传入资源指针以实现
资源的访问。

`Begin_frame_` 类

成员列表：

```
	//默认窗口宽度
	const int begin_frame_width = 800;
	//默认窗口高度
	const int begin_frame_height = 800;
	//地图大小
	int map_width;
	int map_height;

	//残局信息
	std::vector<std::vector<bool>> map;			// 地图状态
	std::vector<std::vector<int>> blockColors;	// 方块颜色

	//资源指针
	Animation* animation;
	IMAGE* background;

	//游戏速度
	int gameSpeed;
	//随机种子
	int randomSeed;
	//游戏难度
	int initialLevel;
	// 存储配置项的键值对
	std::unordered_map<std::string, std::string> configMap;

	//消息结构体
	struct MainMenuMessage {
		bool startGame = false;
		bool quitGame = false;
		bool createConfig = false;
		bool loadConfig = false;
		bool challengeEndGame = false;
		bool createEndGame = false;
	};
	// 主菜单消息循环
	MainMenuMessage menuMsg;
```

方法列表：

```
	/*
	* @brief 程序开始时，仅初始化一次
	* @param animation : 图片资源指针
	*/
	Begin_frame(Animation* animation);
	~Begin_frame();
		/*
	* @brief 初始化游戏主界面
	*/
	void initial();
```

```
    /*
    * @brief 默认参数
    */
    void default_param();

    /*
    * @brief 绘制地图背景
    */
    void inline draw_backgroud();

    /*
    * @brief 绘制菜单
    */
    void inline draw_menu();

    /*
    * @brief 获取消息
    */
    bool getMainMenuMessage(MainMenuMessage& msg);

    /*
    * @brief 加载指定配置文件
    */
    bool loadConfig(const std::string& filename);

    /*
    * @brief 保存当前配置到指定文件
    */
    bool saveConfig(const std::string& filename) const;

    /*
    * @brief 用户交互：新建配置文件
    */
    bool createConfig(std::string &name);

    /*
    * @brief 加载上次使用的配置文件
    */
    bool loadLastConfig();

    /*
    * @brief 保存当前配置为上次使用的配置
    */
    bool saveLastConfig() const;

    /*
    * @brief 获取某个配置项的值
    */
    std::optional<std::string> getConfig(const std::string& key) const;

    /*
    * @brief 设置某个配置项的值
    */
    void setConfig(const std::string& key, const std::string& value);
```

```
        /*
        * @brief 重置为默认配置
        */
        void resetToDefault();

        /*
        * @brief 验证整个配置是否合法
        */
        bool isValid() const;

        /*
        * @brief 验证单个键值对是否合法
        */
        bool validateKeyValue(const std::string& key, const std::string& value) const;

        /*
        * @brief 初始化默认配置
        */
        void initializeDefaults();

        /*
        * @brief 公共方法，用于测试键值对是否合法
        */
        bool testValidateKeyValue(const std::string& key, const std::string& value)
const;

        /*
        * @brief 判断文件是否存在
        */
        bool fileExists(const std::string& filename);

        /*
        * @brief 确认目录存在，如果不存在则创建
        */
        bool createDirectoryIfNotExists(const std::string& path);
```

实现逻辑：
程序开始时，只实例化一个 `Begin_frame`
对象，并进行初始化，显示窗口，之后开始接收消息。根据需求，总共接收六种类型的消息。
每次接收到除了开始游戏和退出游戏之外的有效消息之后，关闭窗口，在控制台内进行配置文件和残局文件的读写操作，读写完成之后，
重新初始化一个窗口，并加载上次启动游戏的配置。
每次启动游戏，主菜单会自动记录启动游戏的配置，将其命名为`last.cofig`文件储存起来，再次初始化的时候会自动读取默认配置文件。
并且，启动游戏时会获取`begin_frame`中已经初始化好的参数，包括地图的大小、关卡难度、初始速度、残局信息。

`Frame`类

成员列表：

```
        //菜单的宽度
        int menu_width;
        //菜单的长度
        int menu_height;

        //界面的宽度
        int frame_width;
        //界面的长度
        int frame_height;

        //地图的宽度，即能容纳的方块数目
        int map_width;
        //地图的宽度，即能容纳的方块数目
        int map_height;

        //储存时间消息
        ExMessage message;

        //W,A,S,D,SPACE各个按键的状态
        bool is_up;
        bool is_down;
        bool is_left;
        bool is_right;
        bool is_space;
        bool is_pause;
        bool is_generate_end_game;
        bool is_over;

        //方块矩阵
        std::vector<std::vector<Block*>> block;

        //当前下落方块组合
        std::vector<std::vector<Block*>> block_group;
        //下一个方块组合
        std::vector<std::vector<Block*>> next_block_group;

        //方块组合字符表示的index
        int block_group_png_index;

        //方块组合图片显示的坐标
        int next_group_block_x_axis;
        int next_group_block_y_axis;

        //玩家得分
        int score;

        //得分坐标
        int score_x_axis;
        int score_y_axis;

        //游戏暂停坐标
        int pause_x_axis;
        int pause_y_axis;
```

```
//速度坐标
int speed_x_axis;
int speed_y_aixs;

//关卡难度坐标
int level_x_axis;
int level_y_axis;

//下落速度
int SPEED;

//随机数种子
int seed;

//关卡数
int level;

//残局信息
//地图状态
std::vector<std::vector<bool>> map;
// 方块颜色
std::vector<std::vector<int>> blockColors;

//主界面指针
Begin_frame* begin_frame;
//图片资源指针
Animation* animation;
//背景照片
IMAGE* background;
//方块组合照片
std::vector<IMAGE*>* block_group_png;

enum blcok_group
{
	S, Z, L, J, I, O, T//方块组合，待定，仍可添加，顺序和加载方块图片的顺序一致
};

//游戏是否正在进行
bool running;
```

方法列表：

```
/*
* @brief 游戏初始化时构造，主函数中调用一次
* @param animation: 资源类指针， begin_frame : 主界面指针
*/
Frame(Animation* animation, Begin_frame* begin_frame);

/*
* @brief 游戏结束后调用，主函数中析构
*/
~Frame();
```

```
        /*
    * @brief 开始游戏
    */
    void game_begin();

    /*
    * @brief 获得玩家操作信息
    * @param message 用于存储信息的结构体
    */
    void get_message(ExMessage& message);


    /*
    * @brief 检查是否可以消除
    */
    void check_line();

    /*
    * @brief 消除满足条件的行，且只下落一次
    * @param row : 行数
    */
    void erase_line(int row);

    /*
    *@brief 刷新游戏界面
    */
    void renew_frame();

    /*
    *@brief 绘制方块
    */
    void draw_block();

    /*
    *@brief 显示得分
    */
    void draw_score();

    /*
    *@brief 显示速度
    */
    void draw_speed();

    /*
    *@brief 显示关卡
    */
    void draw_level();

    /*
    *@brief 显示下一个方块组合的照片
    */
    void draw_block_group_png();

    /*
```

```
    *@brief 控制方块移动：左
    */
    bool moveLeft();
    /*
    *@brief 控制方块移动：右
    */
    bool moveRight();
    /*
    *@brief 控制方块移动：下落
    */
    bool moveDown();
    /*
    *@brief 控制方块移动：下落到最底
    */
    bool moveToLowestPosition();

    /*
    *@brief 旋转方块
    */
    void rotate();

    /*
    * @param targetRow : 目标行， targetColumn : 目标列
    * @brief 检查目标位置是否有碰撞
    * 以行列偏移量（deltaRow, deltaColumn）为参数，用于在移动或旋转方块之前进行检测。
    * 当准备向某个方向（左、右、下）移动或在旋转后变换坐标时，
    * 只需根据新坐标计算出相对于当前坐标的偏移（或新位置），然后调用该函数检查是否有碰撞
    */
    bool checkCollision(int targetRow, int targetColumn);

    /*
    * @brief 绘制地图背景
    */
    void draw_backgroud();

    /*
    * @brief 初始化方块，即生成方块对象
    */
    void initial_block();

    /*
    * @brief 生成方块组合
    */
    void generate_block_group();

    /*
    * @brief 更新block_group
    */
    void rewnew_block_group();

    /*
    * @brief 在方块落地后，将next_block_group传递给block_group
    */
```

```
        void trans_block_group();

        /*
        * @brief 方块组合落地
        */
        void block_group_ground();

        /*
        * @brief 释放方块组合的内存
        */
        void delete_block_group();

        /*
        * @brief 判断是否出现溢出地图上边界，也即游戏结束
        */
        void check_over_map();

        /*
        * @brief 判断游戏结束。当出现方块组合第一次溢出的时候，游戏结束
        */
        void game_over();

        /*
        * @brief 生成残局
        */
        void generate_end_game();

        /*
        * @brief 显示游戏暂停文字
        */
        void draw_pause();

        /*
        * @brief 显示游戏结束文字
        */
        void draw_game_over();
```

实现逻辑：
首先将整个地图初始化为方块对象，并按照行列确定每一个方块的坐标位置，并且额外在地图顶部生成四行，用于生成方块组合使用。
之后生成方块组合，方块组合类型根据给出的随机种子生成，并拥有初始的下降速度，实际的下降速度会随着关卡难度的提高而不断加快。
同时接收来自键盘的消息，可以进行合法的移动和旋转，并排除不合法的输入。
当检测到方块组合落地时，会先将方块组合的信息传递给地图，更新地图上方块的状态，包括颜色和是否存在方块；紧接着检测是否出现溢出，
也即方块出现在地图顶部截止线之上，此时会判断游戏结束，并返回主菜单，等待进一步的操作。
若未出现游戏结束，会先调用检查函数，检查是否出现了可以消除的情况，如果出现了消除，消除行上面的方块会整体向下移动一行，且
再次检测是否会出现消除的情况。该循环会一直执行直到出现无法再次消除。如果在消除的过程中出现了连续消除，会将再次消除的得分翻倍，

翻倍为指数翻倍。在停止消除之后，会根据得分情况，如果得分超过了`关卡难度 * 10`，关卡难度升级，下降速度提高。
游戏帧率为`60`帧，采取双缓冲技术，并引入动态延时优化性能，在刷新时会依次调用需要绘制的信息，保证图层上下合理。

`Block`类

成员列表：

```
	//方块宽度，由方块的图片决定
	const static int block_width = 30;

	//方块高度，由方块的图片决定
	const static int block_height = 30;

	//方块所在排数
	int row;

	//方块所在列数
	int column;

	//该位置是否存在方块
	bool is_block = false;

	//方块颜色
	int color;

	//存储方块图片
	std::vector<IMAGE*>* block_png;
```

成员函数：

```
	/*
	* @param row : 该方块的行序号， column : 该方块的列数序号；均以0开始
	* @brief 生成方块对象
	*/
	Block(int row, int column, std::vector<IMAGE*>* block_png);

	~Block();

	/*
	* @brief 在界面上绘制方块
	*/
	void show();
```

实现逻辑：
每个方块对象具有自己的行列信息，并据此计算出坐标信息。
每次绘制时，会先检测是否是方块，如是，会根据颜色显示不同的方块图片。

`EndGame`类

成员列表：

```
    // 地图宽度
    int mapWidth;
    // 地图高度
    int mapHeight;
    // 初始关卡
    int initialLevel;

    // 地图状态
    std::vector<std::vector<bool>> map;
    // 方块颜色
    std::vector<std::vector<int>> blockColors;

    // 动画资源
    Animation* animation;
```

方法列表：

```
    /*
    * @brief 游戏中生成残局
    */
    EndGame(int map_height, int map_width, int level, std::vector<std::vector<bool>>
map, std::vector<std::vector<int>> blockColors);

    /*
    * @brief 手动创建残局
    */
    EndGame(Animation* animation);

    ~EndGame();

       /*
    * @brief 验证残局是否合法
    */
    bool isValid() const;

    /*
    * @brief 创建残局
    */
    bool createEndGame(std::string& endGameName);

    /*
    * @brief 保存残局到文件
    */
    bool saveToFile(const std::string& filename) const;

    /*
    * @brief 从文件加载残局
    */
    bool loadFromFile(const std::string& filename);

    /*
    * @brief 可视化残局地图
    */
    void visualizeEndGame();
```

实现逻辑：
在`EndGame`类中，实现了残局（自定义）生成、残局加载。在生成时，需要用户手动输入残局的名字；在加载残局时，如果用户输入不合法，会强制重新输入；
当然，也可以随时退出残局，放弃残局的生成或者加载。

小组分工

陈道凡：

1. 构建整个项目框架
2. 实现`Frame`类
3. 实现`Animation`类
4. 实现`Block`类
5. 编写主函数消息处理逻辑

谢宇涵：

1. 实现`EndGame`类
2. 实现`Begin_frame`类
3. 编写主函数处理逻辑
4. 完善控制台界面和游戏界面切换