2021 年 6 月 14 日

技术报考

1　技术要求

1.1　介绍

在这次的大作业，我选择了游戏类的题目之一，就是“太鼓达人”的游戏(题目 6.22)。下面是题目的基本要求:

- **实现太鼓达人游戏。**
- **能够使用键盘按照背景音乐的节奏进行击鼓，不同按键对应不同颜色的鼓面。**
- **鼓面从左向右移动至打击处，能够根据打击时刻的准确度进行计分并计算连击数。**
- **至少要做两首音乐。**

1.2　题目分析

因为这个是一个游戏，我打算使用一个 C++语言的游戏框架和面向对象程序设计的技术。现在，我们看题目的每个具体的要求：

1. **实现太鼓达人游戏。**

   为了实现这个游戏，我打算创作任何游戏的基本要求, 就是创作主游戏菜单画面，游戏开始画面，游戏玩法画面，和游戏玩法结束画面。游戏也一定包括适当和吸引人的图形 （游戏精灵和图像）。

2. **能够使用键盘按照背景音乐的节奏进行击鼓，不同按键对应不同颜色的鼓面。**

   其实，为了实现“使用键盘按照背景音乐的节奏进行击鼓”的要求有两个方法: 使用迷笛（*MIDI*）技术 或者使用时间戳的技术。这次，我选择时间戳的技术来按照音乐和节奏的联系。然后，为了实现“不同按键对应不同颜色的鼓面”的要求，我打算使用在游戏框架的

KeyEventListener 的 API(事件监听器的应用程序接口)来接受键盘的任何具体输入。然后，不同颜色的鼓面能使用不同的游戏精活（*sprites*）。

3. **鼓面从左向右移动至打击处，能够根据打击时刻的准确度进行计分并计算连击数。**

   为了实现“鼓面从左向右移动至打击处”的要求，我打算使用刷怪(*spawn*)的函数来创作新的鼓面，然后使用移动函数，来连续的更新鼓面的位置。关于时刻的准确度， 在界面窗口界面会有一个目标线（无形的）。当球员在打击的时候，代码会使用一个函数来措施和计算鼓面位置和目标线之间的距离。不同的距离的范围对应具体的积分。

4. **至少要做两首音乐**

   准备两个音乐文件和游戏玩法。

在这个项目，我选择用 SFML C++ 的游戏框架, 因为它已经包括所有的基本图形和其他游戏处理。

2 游戏设计

2.1 游戏代码基本结构

每个游戏的代码能分成有些基本的重要部分：

1. **游戏窗口连续循环**

   在游戏的代码里面，我们平时能看到一个普通循环，写着“`while(window->isOpen())`”。这个是游戏无限的循环，就是让游戏连续地通过改变。在这个循环，代码会更新游戏的数据和输出更新的结果，并且渲染任何图形。在循环里也能接受输入。

2. **更新 （Update）**

   每个游戏有主更新函数，就是为了更新任何在游戏里的改变。

3. **渲染 （Render）**

   每个游戏也有主渲染的函数，就是为了连续地渲染任何游戏的图形，而且渲染更新函数结果的图形。

4. **事件轮询 （Event Polling）**

   最后，每个游戏有主事件轮询的函数，就是为了按照用户的输入（键盘事件, 鼠标事件等等）。

这四个原则组成任何游戏的基本结构。它们互相影响，并且一起工作的。在太鼓达人游戏，一个基本的例子是鼓面的移动。为了指定不同颜色鼓面的位置，我们有`setPosition()`的函数。在无限的循环，`setPosition()`的函数是放在“更新鼓面”的函数， 所以这个函数连续第指定鼓面的动态的位置。当鼓面的位置变的时候，“渲染鼓面”的函数能渲染鼓面的新位置和图形。

最后，事件轮询函数可能接受为了打击鼓面的键盘输入。当接受输入后，更新鼓面的函数能删除被打击的鼓面数据。然后，渲染鼓面的函数停着渲染被删除的鼓面数据。

2.2 类设计

在我的代码里，我打算生成四个主头文件：State.h （状态），Game.h （游戏），Graphics.h （图形），GameData.h （游戏数据）。下面是每个头文件的内容规划和具体功能。

2.2.1 游戏头文件

游戏头文件只包括游戏的游戏类，Game class，就是代码的主类。游戏类的主要功能是控制无限循环的状态，按照用户的任何输入（键盘，鼠标等），和包含游戏的主更新和渲染函数。下面是这个类的内容。

SFML 框架的变量都有“sf::”就是 SFML 的主类。

变量		
名	变量类型	功能
window	私有，sf::RenderWindow *	全游戏的主窗口界面。
videoMode	私有，sf::VideoMode	为了指定窗口的基本配置，例如指定尺寸维度是 （宽度 x 高度）： 1440 x 720px。
sfEvent	私有，sf::Event	输入事件
dt	私有，float	deltaTime 的时间，为不同的 CPU 保持恒定的帧率
dtClock	私有，sf::Clock	deltaTime 钟

函数		
名	返回类型	功能
Game()	公共	构造函数，进行所有初始化的函数
virtual ~Game()	公共	删除
InitWindow()	私有，void	初始化 videoMode 和 window 的变量。
InitStates()	私有，void	初始化状态，为 states 的堆栈推新的状态。（状态的描述在下一个页）
updateDt()	公共，void	更行 deltaTime 值
updateSFMLEvents()	公共，void	事件轮询的函数
update()	公共，void	主更新函数，所有状态的更新函数在这里进行的。

render()	公共，void	主渲染函数，所有渲染的函数在这里进行的。
run()	公共，void	进行游戏，使用无限循环“while(window.isOpen())”

2.2.2 状态头文件

因为这个游戏使用几个画面（*screens*）的技术，这些画面能分成有些具体的状态。这部分使用面向对象程序设计的继承性。首先，我制作一个主状态类，叫 State class。为了理解具体 State 类的逻辑，下面是 State 类的内容：

变量		
名	变量类型	功能
states	std::stack<State*>*	每个状态的具体类会有 state 堆栈的指针，因为每个状态类以后会推其他的状态类。
window	sf::RenderWindow*	这个是窗口的指针，为了渲染的函数
quit	bool	退出状态类的布尔值
mousePosWindow	sf::Vector2i	窗口实时鼠标位置，像素格式
mousePosView	sf::Vector2f	窗口实时鼠标位置，坐标格式，（这个变量对按钮功能很重要）

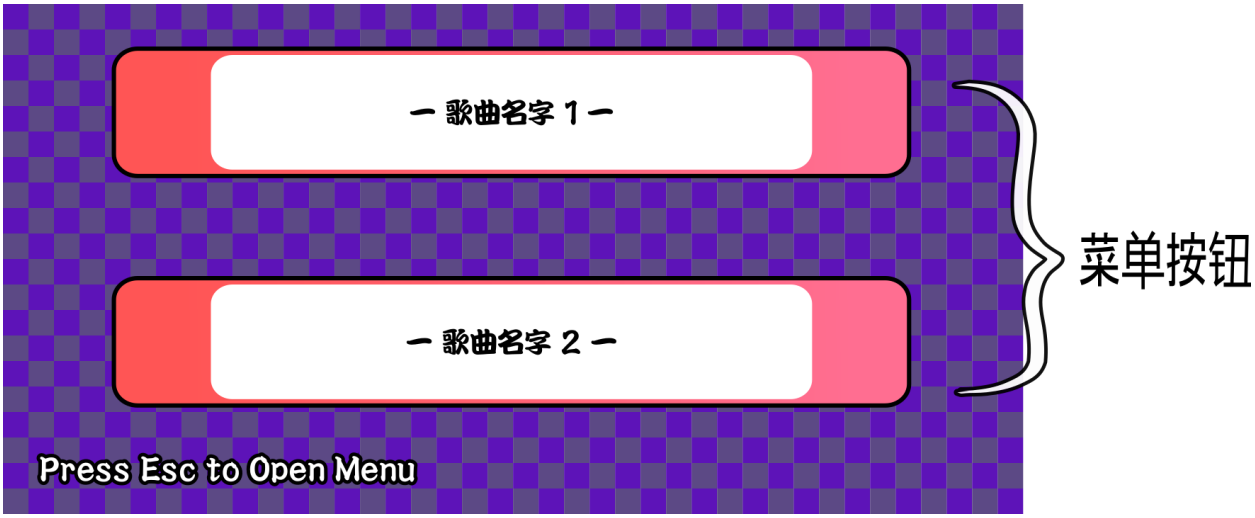
函数		
名	返回类型	功能
State(window, states)	公共	初始化主窗口和 videoMode 的指针，和其他的变量
virtual ~State()	公共	删除
getQuit() const	公共，const bool&	返回推出布尔值
endState() = 0	公共，virtual void	表示在控制窗口的推出状态提醒
updateMousePosition()	公共，virtual void	更新mousePosView的值，就按照 mousePosWindow 的数据，然后把像素格式变成坐标格式，mousePosView。
updateKeybinds(dt, eventType) = 0	公共，virtual void	接受从 Game class 的事件类型，然后就更新事件轮询的结果
update(dt) = 0	公共，virtual void	状态类的主更新类，其他的具体更新函数都放在这里。
render(target = nullptr) = 0	公共，virtual void	状态类的主渲染类，其他的具体渲染函数都放在这里。

从这个类，使用继承性，我们生成了四个派生类：

1. StartPageState class : public State

上面是开始画面的设计，因此游戏第一次开始就表现这个窗口。这个状态是从 `Game class` 的 `InitStates()`函数推到堆栈的。用户能输入键盘 “Enter”，然后代码会到堆栈推下一个的状态，就是游戏菜单，`MenuState class.`

2. MenuState class : public State



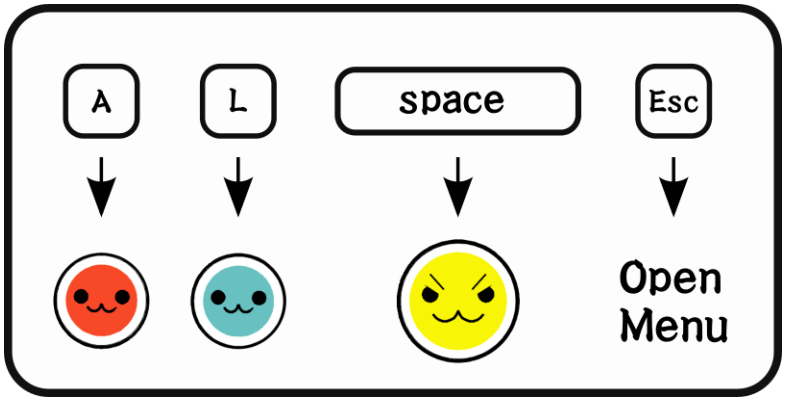
上面是菜单状态的设计。题目的要求是至少有两个歌曲，所以这里我就准备了两个歌曲的按钮。按钮有另外的类叫 `MenuButton class.` 这些按钮有使用枚举的变量，为了按照按钮的状态，分成三个状态：闲态 (BTN_IDLE)，悬停态 (BTN_HOVER)，和按态(BTN_PRESSED)。

按钮逻辑代码：

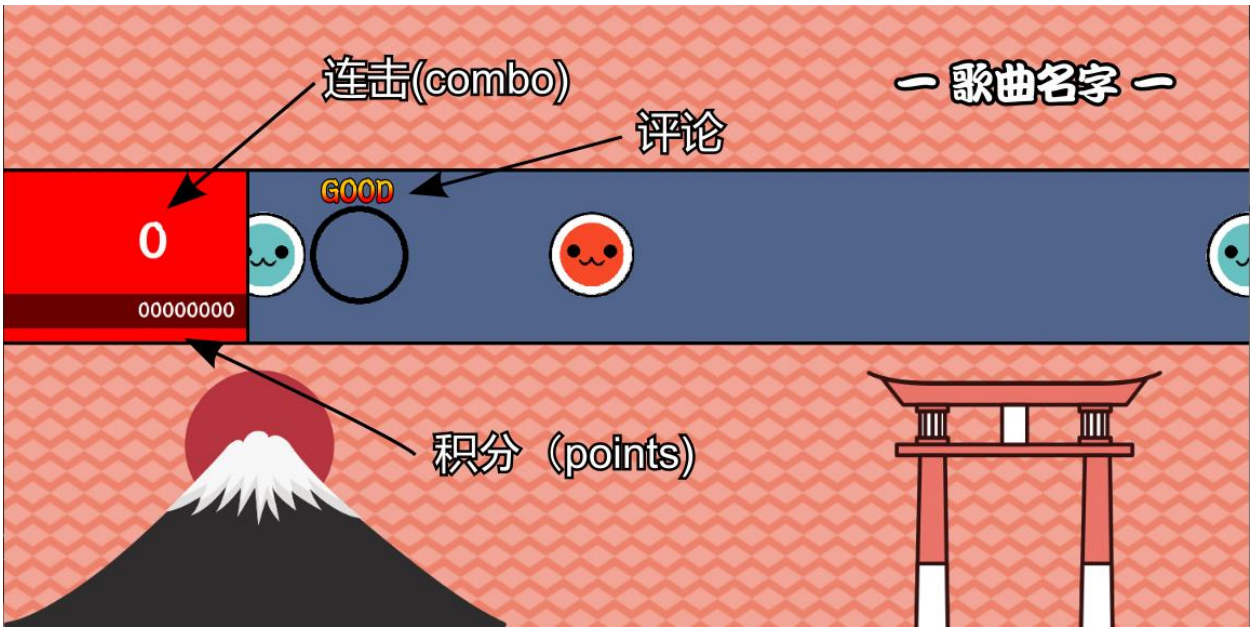
```
void MenuButton::update(const sf::Vector2f mousePos)
{
	if (this->sprite.getTexture() != NULL) {
		this->buttonState = BTN_IDLE;

		if (this->sprite.getGlobalBounds().contains(mousePos)) {
			//Hover
			this->buttonState = BTN_HOVER;

			if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
				this->buttonState = BTN_PRESSED;
			}
		}

		switch (buttonState)
		{
		case BTN_IDLE:
			this->sprite.setColor(sf::Color::Color(100, 100, 100, 255));
			if (this->tempMusic.getStatus() == sf::Music::Playing) {
				this->tempMusic.stop();
			}
			break;
		case BTN_HOVER:
			this->sprite.setColor(sf::Color::Color(255, 255, 255, 255));
			if (this->tempMusic.getStatus() == sf::Music::Stopped) {
				this->tempMusic.setPlayingOffset(sf::seconds(90));
				this->tempMusic.play();
			}
			break;
		case BTN_PRESSED:
			this->sprite.setColor(sf::Color::Color(255, 255, 255, 255));
			this->tempMusic.stop();
			break;
		default:
			break;
		}
	}
}
```

所以，这个代码是检查鼠标坐标是否在按钮里面。如果是的话，按钮状态指定为悬停态。同时，如果用户有鼠标左键单击，按钮的状态变成按态。每个状态都有自己的指定。悬停态会播放音乐的片段，然后按态能推 `GameState class` 到 `State` 的堆栈。

下面是玩法小手册的图形设计：



3. GameState class : public State



上面是玩法状态的普遍表示。鼓面会从左向右移动的，然后有一个“目标圆”，就是用户打击鼓面的地方。在目标圆的中间有一个无形的中线，为了措施鼓面打击的时刻准确度。

- 如果打击很精准，积分增加 750 分。如果打击挺精准，积分增加 500 分。最后，如果打击不精准，积分增加 250 分。
- 每个准确度的类型能返回三个评论：“GOOD”，“BAD”，和“OK”。
- 连击数连续地增加，但如果有鼓面无被打击通过了目标圆，连击数重置为 0。

下面是更新鼓面的代码，包括移动鼓面和决定那一个鼓面是在目标圆里：

```
void GameState::updateTaps()
{
	/**
		@return void

		//updating timer for spawning
		//spawn and reset the timer
	*/
if (tapsIndex < midiBeats.size())
{

	if ((double)this->spawnClock.getElapsedTime().asSeconds() >
	this->midiBeats[tapsIndex].timeStamp - this->pause_time)
	{
		this->pause_time = 0;
		this->spawnTap(midiBeats[tapsIndex]);
		this->tapsIndex++;
		this->spawnClock.restart();
	}
}
```

```
//Move the taps sideways
for (int i = 0; i < taps.size(); i++) {

	//Move the taps sideways
	this->taps[i].tap_sprite.move((float)(TAP_SPEED * -1), 0.f);

	//checking if any taps are intersecting with the target tap
	if
	(this->targetTap.getGlobalBounds().intersects(this->taps[i].tap_sprite.getGlobalB
	ounds()))
	{
		this->intersecting[i] = true;
		this->intersectObjRef = i;
		if (gameStatus == preStart)
			gameStatus = Running;
	}
	else
	{
		if (this->comboStatus == true && this->intersecting[i] == true) {
			if (this->combo > this->max_combo)
				this->max_combo = this->combo;
			this->combo = 0;
			this->comboStatus = false;
		}
		this->intersecting[i] = false;
	}

	//Erase taps after exceeding boundary
	if (this->taps[i].tap_sprite.getPosition().x < this->eraseLine.getPosition().x -
100)
	{
		this->taps.erase(this->taps.begin() + i);
	}

}

}
```

代码的描述是在下面：

- 为了保存鼓面和跟目标圆交叉状态，我是用两个矢量数据 `std::vector<BeatTaps>` (使用自己制作的鼓面类)和 `std::vector<bool>` (布尔值)来按照鼓面状态。
- 使用钟变量来跟踪生成新鼓面的时间。（`sf::clock spawnClock`）
- 在玩法状态的窗口，我有指定无形的线，就是为了删除通过目标圆的鼓面。

```
if (this->gameStatus == Running && !this->taps.empty() && !this->intersecting.empty())
{

bool left = sf::Keyboard::isKeyPressed(sf::Keyboard::A) &&
this->intersecting[intersectObjRef] == true && this->taps[intersectObjRef].info.type ==
BeatType::LEFT;

bool right = sf::Keyboard::isKeyPressed(sf::Keyboard::L) &&
this->intersecting[intersectObjRef] == true && this->taps[intersectObjRef].info.type ==
BeatType::RIGHT;

bool both = sf::Keyboard::isKeyPressed(sf::Keyboard::Space) &&
this->intersecting[intersectObjRef] == true && this->taps[intersectObjRef].info.type ==
BeatType::BOTH;
```

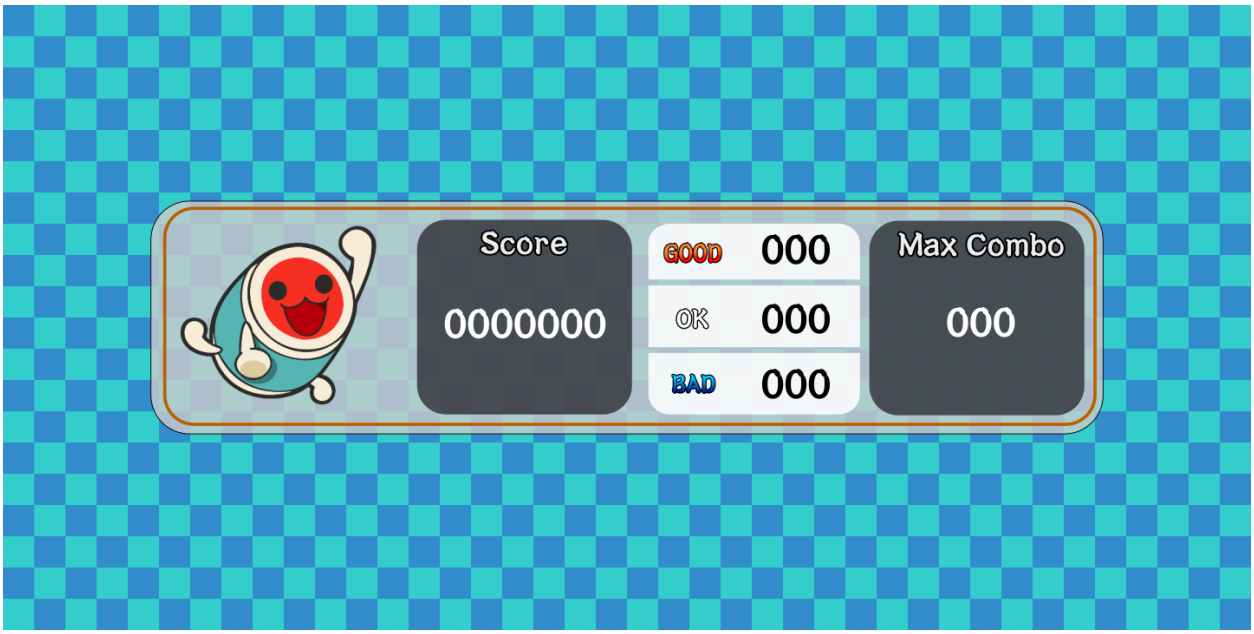
```
        if (left || right || both )
        {
        this->points += this->pointAlloc(this->taps[intersectObjRef].tap_sprite);
        this->taps.erase(this->taps.begin() + intersectObjRef);
        this->intersecting.erase(this->intersecting.begin() + intersectObjRef);
        this->combo++;

        if (!this->comboStatus)
              this->comboStatus = true;
        }
}
```

代码的描述是在下面：

- left，right，和 both 的布尔值对应不同颜色鼓面。
- 假如布尔值之一是 true 的, 积分会增加，使用 pointAlloc 的函数 (接受准确性类型的参数，返回分数)，然后删除被打击的鼓面数据（鼓面类和交叉布尔值的物体）。
- 增加连击数；把连击的布尔值变成 true。

最后，音乐播放结束后就在堆栈推了玩法结束状态的物体。

4. GameEndState class : public State



上面是玩法结束的状态画面。这个画面会使用小的动画片来表示分数。如果表示完后，回去到菜单画面。

2.2.3 图形头文件

这头文件有包含三个类：按钮类，鼓面时间戳类，和玩法状态的图形类。

- 按钮类的内容和函数逻辑基本上在 `MenuState` 类的代码描述相似的。
- 鼓面时间戳类（`Beats class`）只包括两个变量：
    - 鼓面类型：指定鼓面的类型 （使用枚举）
    - 时间戳：指定鼓面生成时间戳
- 玩法状态的图形类是为了指定静态图形。

2.2.4 游戏数据头文件

这头文件包含两个类：音乐玩法数据类，分数类。

- 音乐玩法数据类 （`GameData class`）有包括：
    - 音乐名字，歌手
    - 矢量数据类型：`std::vector<Beats>`
        - 保存音乐鼓面时间戳的数据。
    - 音乐文件路径
- 分数类包含：
    - 连击数，积分数
    - ‘OK’, ‘GOOD’, ‘BAD’ 的评论次数。

3 具体特殊技术

3.1 游戏框架

3.1.1 用过技术小解释

为了实现游戏，我用了 SFML C++ 语言的框架。下面是我常用的 SFML 具体技术和它们的具体功能：

SFML 技术	具体功能
画面，窗口， 事件	
`sf::RenderWindow`	可作为 2D 绘图目标的窗口
`sf::VideoMode`	定义视频模式（宽度、高度），用于设置窗口
`sf::Event`	定义系统事件及其参数, 处理键盘和鼠标的输入
`sf::Keyboard`	
`sf::Mouse`	
图形	
`sf::RectangleShape`	表示矩形的特殊形状, 也能表示线
`sf::CircleShape`	代表圆的特殊形状, 例如表示目标圆
`sf::Texture`	可用于绘图的图形卡上的图像
`sf::Sprite`	纹理的可绘制表示，具有自己的变换、颜色等. (`setTexture` 函数指定图形)
文本	
`sf::Font`	用于加载和操作字符字体的类
`sf::Text`	可以绘制到渲染目标的图形文本
声音、音乐	
`sf::Music`	从音频文件播放的流式音乐,为了播放音乐
`sf::SoundBuffer`	存储定义声音的音频样本

sf::Sound	可以在音频环境中播放的规则声音，使用 setBuffer 函数指定声音内容，为了提供音效，例如打击鼓面的不同音效
钟	
sf::Clock	测量经过时间的实用程序类。
2D 坐标处理，向量	
sf::Vector2i	用于操作二维向量的实用程序模板类, f 是 float, i 是 int
sf::Vector2f	

3.1.2 SFML 具体应用

- **窗口和无限循环**

  SFML 框架提供了一个 2D 窗口和一个 Window.isOpen() 函数，可以实时更新和渲染各种 2D 形状或精灵，方便加载图形，而 Window.draw() 绘图函数允许绘制 2D 图形 直接在 2D 窗口上。

- **物体定位**

  SFML 中从 sf::Shape 类和 sf::Sprite 类派生的所有类或对象都有 setPosition()（定位函数）、setSize()（大小调整函数）和 setFillColor()（填充颜色函数），可用于确定不同 2D 形状的特征。 但是，某些定位必须手动完成，例如在中心对齐 2D 形状。 使用 SFML 居中文本的示例代码如下：

```
text.setPosition(sf::Vector2f
 (
 sprite.getPosition().x + sprite.getGlobalBounds().width / 2 - text.getGlobalBounds().width / 2,
 sprite.getPosition().y + sprite.getGlobalBounds().height / 2 - text.getGlobalBounds().height / 2
 )
);
```

- **文本**

  sf::Text 类有各种方便的函数，如 setString()（设置文本对象的字符串内容）、setCharacterSize()（设置文本对象的字体大小）和 setFont()（设置文本对象的字体类型）。 sf::Text 类必须与 sf::Font 类一起使用，其中字体类型通常为 .ttf 格式。

- **音乐，声音**

sf::Music 类有一个播放功能，由 play()（播放音乐）、pause()（暂停音乐）和 stop()（停止音乐）组成。此外，sf::Music 有一个函数叫 setPlayingOffset()，它允许从任何时间戳播放音乐。在这种情况下，我使用此功能在鼠标指针悬停在菜单按钮上时播放音乐的中部。除此之外，sf::Sound 和 sf ::SoundBuffer 类在播放短音效时很有用，因为它们是即时处理的。

- **钟**

sf::Clock（时钟类型）用于这个游戏的各种应用，特别是鼓面的生成过程。 时钟在构造时自动开始运行，并且只能通过 restart()（重新启动函数）进行重置。 这个类没有暂停功能。由于运行时间不确定，我们不能使用相等比较（“==”），而是使用“大于”比较。 示例代码如下：

```
if ((double)spawnClock.getElapsedTime().asSeconds() > midiBeats[tapsIndex].timeStamp -
pause_time)
{
	pause_time = 0;
	spawnTap(midiBeats[tapsIndex]);
	tapsIndex++;
	spawnClock.restart();
}
```

- **键盘事件**

SFML 有一个名为 sf::Keyboard::isPressed() 的函数，它返回一个特定的键码是否被点击的布尔值。 示例代码如下：

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::A) && !this->red_sound_bool)
{
	this->red_sound.play();
	this->red_sound_bool = true;
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::L) && !this->blue_sound_bool)
{
	this->blue_sound.play();
	this->blue_sound_bool = true;
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)
&& !this->gold_sound_bool)
{
	this->gold_sound.play();
	this->gold_sound_bool = true;
}
```

3.2 C++另外技术

3.2.1 STL

- *堆栈（stack）*

```
//CODE SNIPPETS

this->states.push(new StartPageState(this->window, &this->states));

this->states->push(new GameEndState(this->window, this->states, temp));


//GAME CLASS UPDATE FUNCTION

if (!this->states.empty())
{
	this->states.top()->update(this->dt);

	if (this->states.top()->getQuit())
	{
		this->states.top()->endState();
		delete this->states.top();
		this->states.pop();
		std::cout << "endState..." << std::endl;
	}
}
```

如前所述，堆栈数据结构用于跟踪状态对象。首先，主游戏类会推开始状态对象。然后，其他的状态对象会互相推到堆栈。主游戏使用 `stack.top()`，只把堆栈的最佳对象进行更新和渲染函数。状态结束后，主游戏类会使用 `stack.pop()`，单出和删除状态，回去以前的状态。

- *矢量（vector）*

矢量数据结构用于存储各种类型的数据，如时间戳、鼓面信息等，如上一节所述。 很方便，因为 `std::vector` 是一个动态结构，例如鼓面信息的 `vector` 可以动态伸缩。 因此，它可以节省空间并有助于优化游戏的处理速度。

- *映射（map）*

`Map` 通常用于使数据搜索几乎是即时的，因为您可以使用任何标签索引数据。 例如，`std::map<std::string, MenuButton* button>` 意味着您可以创建一组具有特定名称作为其索引的按钮。 在这个 游戏中，我将它用于代码组织目的。

3.2.2 C++/面向对象程序设计的特性

- 继承性

  许多类利用继承原则，主要是 States 类，因为有多个派生状态类来表示不同的游戏屏幕，例如开始屏幕和游戏屏幕。 由于每个状态都必须有一个更新函数、渲染函数和一个事件函数，这些函数在主状态类中被声明为虚函数（virtual void function() = 0）。 因此，必需的函数总是在派生类中声明。 游戏中使用继承的其他类是按钮类。

- 封装性

  大多数类还使用了封装，例如私有类型、公共类型和受保护的变量。 例如 State 类有一个受保护的类型变量，即 MousePosView（窗口实时鼠标位置，坐标格式），可以被其派生类中的函数使用。 在这种情况下，MenuState（菜单状态类）中的 MenuButton（菜单按钮类）使用 MousePosView 变量来确定其按钮状态（您可以在第 5 页中看到此应用程序）。

- 指针

  一些变量使用指针，以便其他派生类可以访问指向一个公共 2D 目标窗口的指针，例如 sf::RenderWindow* 和 States*。 对于 RenderWindow，其他类可以使用 render 方法通过指针访问直接在窗口上绘制。 对于状态，派生的状态类可以执行 push() 或 pop() 函数（堆栈推送或堆栈弹出）以在游戏中的不同屏幕之间导航。

4 游戏的结果

我有准备了一个游戏演示视频，能参考一下。游戏的演示文件名是 “Taiko No Tatsujin v2.exe”。

游戏的成功标准	
每个题目的要求已经完成。	
有提供了额外功能，包括按钮的复杂性，动画片 （淡入淡出，按钮状态，评委动画片等）。	
提供游戏的几个屏幕，让用户真的享受玩太鼓达人游戏的感觉。	
可以改进的部分	改进方式
游戏缺少数据保存功能，用户可以在其中保存播放的每首音乐的高分。	学 Window config file (窗口配置) , 或者使用 fstream （C++ 文件处理的头文件库）, 把游戏的数据保存在别的文件
鼓面的节奏和音乐在不同的电脑上可能不会完全对应，因为节奏使用了时间戳。 虽然我已经使用了有助于保持节奏的增量时间，但不同的计算机可能有不同的处理速度。 因此，鼓面的速度可能会滞后或移动得更快。	使用 MIDI （迷笛）技术来指定节奏的高准确性。
图形仍然可以改进以类似于原始的太鼓达人的游戏。	使用更高级的游戏框架，例如 C# XNA 游戏框架, 提高形状处理的效率

5 总结

总之，我对最终的游戏产品非常满意。 虽然游戏并不完美，但它功能齐全，其他人可以玩，只要他们有一台装有 Windows 操作系统的电脑。 老实说，这是我迄今为止在清华最有趣的项目之一，因为这是我第一次在不使用游戏引擎（例如 Unity 或 Unreal Engine）的情况下制作游戏。 C++ SFML 游戏框架是一个极其完整的库，可用于制作任何 2D 游戏。 未来，我希望学习更高级的游戏框架，包括 XNA 和 Monogame。 即使我不在计算机科学系，这个项目也让我更喜欢学习编程。 因此，我将继续学习更多有关游戏开发和其他内容的知识，例如 Web 开发、应用程序开发和机器语言。

6 引文

SFML 的文件：https://www.sfml-dev.org/documentation/2.5.1/