1. 检查按键状态

通常在游戏中，你可能会对特定的已知按键以及检测它们何时被按下或释放感兴趣。你可以使用 ButtonInput<KeyCode> 资源来检查特定按键。

使用 .pressed(…)/.released(…) 来检查按键是否被按住 只要按键处于相应状态，这些方法每帧都会返回 true。 使用 .just_pressed(…)/.just_released(…) 来检测实际的按下/释放这些方法仅在按下/释放发生的帧更新时返回 true。

要遍历当前按住的任何按键，或已按下/释放的按键：

```
fn move_player(
    mut player: Query<&mut Transform, With<Player>>,
    keyboard: Res<ButtonInput<KeyCode>>,
    time: Res<Time>,
) {
    let Ok(mut transform) = player.get_single_mut() else {
        return;
    };
    let speed = 5. * time.delta_seconds();
    let mut velocity = Vec3::ZERO;
    if keyboard.pressed(KeyCode::ArrowUp) {
        velocity += transform.forward() * speed;
    }
    if keyboard.pressed(KeyCode::ArrowDown) {
        velocity += transform.back() * speed;
    }
    if keyboard.pressed(KeyCode::ArrowLeft) {
        velocity += transform.left() * speed;
    }
    if keyboard.pressed(KeyCode::ArrowRight) {
        velocity += transform.right() * speed;
    }
    transform.translation += velocity;
}

fn clear_text(mut text: Query<&mut Text>, keyboard:
Res<ButtonInput<KeyCode>>) {
    if keyboard.just_pressed(KeyCode::Enter) {
        if let Ok(mut text) = text.get_single_mut() {
            text.sections[0].value.clear();
        }
    }
}
```

2. 运行条件

另一种工作流程是为你的系统添加运行条件，使它们仅在适当的输入发生时运行。

强烈建议你编写自己的运行条件，以便你可以检查任何你想要的内容，支持可配置的绑定等。

对于原型设计，Bevy 提供了一些内置的运行条件：

```
toggle_game_state.run_if(input_just_pressed(KeyCode::Space)),
```

3. 键盘事件

要获取所有的键盘活动，你可以使用 KeyboardInput 事件：

```
fn input_text(mut text: Query<&mut Text>, mut
keyboard_reader: EventReader<KeyboardInput>) {
    let Ok(mut text) = text.get_single_mut() else {
        return;
    };
    for keyboard in keyboard_reader.read() {
        if let ButtonState::Released = keyboard.state {
            continue;
        }
        match &keyboard.logical_key {
            Key::Backspace => {
                text.sections[0].value.pop();
            }
            Key::Character(string) => {
                if string.chars().any(char::is_control) {
                    continue;
                }
                text.sections[0].value.push_str(&string);
            }
            _ => continue,
        }
    }
}
```

4. 物理键码 vs. 逻辑键

当按下一个键时，事件包含两个重要的信息：

- KeyCode，它始终代表键盘上的特定键，无论操作系统布局或语言设置如何。
- Key，它包含操作系统解释的键的逻辑含义。

当你想实现游戏机制时，你应该使用 KeyCode。这将为你提供可靠的按键绑定，包括对配置了多个键盘布局的多语言用户。

当你想实现文本/字符输入时，你应该使用 Key。这可以为你提供 Unicode 字符，你可以将其附加到你的文本字符串中，并允许用户像在其他应用程序中一样输入。

如果你想处理键盘上具有特殊功能键或媒体键的键盘，这也可以通过逻辑键来完成。

5. 文本输入

以下是如何将文本输入实现到字符串中的一个简单示例（这里存储为本地资源）。

```
fn input_text(mut text: Query<&mut Text>, mut
keyboard_reader: EventReader<KeyboardInput>) {
    let Ok(mut text) = text.get_single_mut() else {
        return;
    };
    for keyboard in keyboard_reader.read() {
        if let ButtonState::Released = keyboard.state {
            continue;
        }
        match &keyboard.logical_key {
            Key::Backspace => {
                text.sections[0].value.pop();
            }
            Key::Character(string) => {
                if string.chars().any(char::is_control) {
                    continue;
                }
                text.sections[0].value.push_str(&string);
            }
            _ => continue,
        }
    }
}
```

注意我们如何为 Backspace 和 Enter 键实现特殊处理。你可以轻松地为其他在你的应用程序中有意义的键（如箭头键或 Escape 键）添加特殊处理。

为我们的文本生成有用字符的键以小的 Unicode 字符串形式出现。在某些语言中，每次按键可能会有多个字符。

注意：为了支持使用复杂文字语言（如东亚语言）或使用手写识别等辅助方法的国际用户的文本输入，除了键盘输入外，你还需要支持 IME 输入。