Input Handler

(Controller)

TPV2

Samir Genaim

¿Qué es el Patrón Input Handler?

El Patrón Input Handler

Es una abstracción sobre el manejo de entrada. El objetivo es agregar todos los eventos de entrada en un sólo objeto y proporcionar un punto de acceso global a ese objeto para consultar los eventos, etc.

El bucle principal de un juego ...

El bucle principal del juego tiene la siguiente forma

```
while (!exit_) {
    ...
    // handle input
    while (SDL_PollEvent(&event)) {
        ...
        for (auto &o : objs_) o->handleInput(event);
    }
    ...
}
```

El problema con este diseño:

1. Para cada evento llamamos a handleInput de cada GameObject, aunque no los usa
2. Si queremos proporcionar otro tipo de entrada, tenemos que cambiar el método handleInput de GameObject.

El objetivo es quitar el parámetro de handleInput y proporcionar otra manera de manejar la entrada: usando InputHandler

La clase InputHandler

En nuestro diseño, InputHandler es una clase singleton

- Mantiene un estado con todos los eventos de entrada en la última iteración del bucle principal
  - Tiene un método update(SDL_Event& event) para pasarle un evento (para actualizar su estado)
  - Tiene un método clearState() para borrar el estado anterior (los eventos en la última iteración)
- Tiene métodos para consultar el estado (teclas, raton, etc) que se puede usar desde cualquier parte que tiene referencia al InputHandler

InputHandler: El bucle principal

Borra el estado anterior y pasa todos los eventos de esta iteración al InputHandler

```
while (!exit_) {
    ...
    InputHandler::instance()->clearState();
    while (SDL_PollEvent(&event))
        InputHandler::instance()->update(event);
    ...
    for (auto &o : objs_) o->handleInput();
    for (auto &o : objs_) o->update();
    for (auto &o : objs_) o->render();
    ...
}
```

inputHandle de GameObject no recibe el evento. Sólo una llamada por objeto, como el update y el render

InputHandler: ejemplo de uso

```
void Hero::handleInput(const SDL_Event& event) {
    ...
    if (event.type == SDL_KEYDOWN) {
        if (event.key.keysym.sym == SDLK_a) {
            ...
        } ...
    }
}
```

sin InputHandler

```
void Hero::handleInput() {
    ...
    if (InputHandler::instance()->isKeyDown(SDLK_a)) {
        ...
    } ...
}
```

con InputHandler

La Clase InputHandler: I

```
class InputHandler : public Singleton<InputHandler> {
...
public:
    ...
    inline void clearState();
    inline void update(const SDL_Event &event);

    // keyboard
    inline bool keyDownEvent();
    inline bool keyUpEvent();
    inline bool isKeyDown(SDL_Scancode key);
    inline bool isKeyDown(SDL_Keycode key);
    inline bool isKeyUp(SDL_Scancode key);
    inline bool isKeyUp(SDL_Keycode key);

    // mouse
    inline bool mouseMotionEvent();
    inline bool mouseButtonEvent();
    inline const std::pair<Sint32, Sint32>& getMousePos();
    inline bool getMouseButtonState(MOUSEBUTTON b);
    ...
};
```

Para actualizar el estado

Para consultar el estado del teclado

Para consultar el estado del ratón

```
enum MOUSEBUTTON : uint8_t {  LEFT = 0, MIDDLE = 1, RIGHT = 2 };
```

La Clase InputHandler: II

```
class InputHandler {
   ...
private:
    ...
   bool isKeyUpEvent_;
   bool isKeyDownEvent_;
   bool isMouseMotionEvent_;
   bool isMouseButtonEvent_;
   std::pair<Sint32, Sint32> mousePos_;
   std::array<bool, 3> mbState_;
   const Uint8 *kbState_;
};
```

Booleans para indicar el tipo de evento ...

La posición del mouse y el estado de los botones

kbState_[SDL_SCANCODE_A] es 0/1 depende del estado de la tecla A. Es un array de SDL

La Clase InputHandler: III

```
InputHandler() {
    clearState();
    kbState_ = SDL_GetKeyboardState(0);
}

inline void clearState() {
    isKeyDownEvent_ = false;
    isKeyUpEvent_ = false;
    isMouseButtonEvent_ = false;
    isMouseMotionEvent_ = false;
    for (int i = 0; i < 3; i++) {
        mbState_[i] = false;
    }
}
```

Pedimos a SDL el array que tiene los estados de las teclas — solo una vez

Borra el estado actual. El array kbState_ lo modifica SDL

La Clase InputHandler: VI

```
inline void update(const SDL_Event &event) {
    switch (event.type) {
    case SDL_KEYDOWN:
        onKeyDown(event);
        break;
    case SDL_KEYUP:
        onKeyUp(event);
        break;
    case SDL_MOUSEMOTION:
        onMouseMotion(event);
        break;
    case SDL_MOUSEBUTTONDOWN:
        onMouseButtonChange(event, true);
        break;
    case SDL_MOUSEBUTTONUP:
        onMouseButtonChange(event, false);
        break;
    ...
    }
}
```

depende de evento, invocamos a métodos correspondientes para actualizar el estado

La Clase InputHandler: V

```
inline void onKeyDown(const SDL_Event&) {
    isKeyDownEvent_ = true;
}

inline void onKeyUp(const SDL_Event&) {
    isKeyUpEvent_ = true;
}

inline void onMouseMotion(const SDL_Event &event) {
    isMouseMotionEvent_ = true;
    mousePos_.first = event.motion.x;
    mousePos_.second = event.motion.y;


}
inline void onMouseButtonChange(const SDL_Event &event, bool isDown) {
    isMouseButtonEvent_ = true;
    switch (event.button.button) {
    case SDL_BUTTON_LEFT:
        mbState_[LEFT] = isDown;
        break;
    case SDL_BUTTON_MIDDLE:
        ...
}
```

La Clase InputHandler: IV

```
inline bool keyDownEvent() {
    return isKeyDownEvent_;
}
inline bool keyUpEvent() {
    return isKeyUpEvent_;
}
inline bool isKeyDown(SDL_Scancode key) {
    return keyDownEvent() && kbState_[key] == 1;
}
inline bool isKeyDown(SDL_Keycode key) {
    return isKeyDown(SDL_GetScancodeFromKey(key));
}
inline bool isKeyUp(SDL_Scancode key) {
    return keyUpEvent() && kbState_[key] == 0;
}
inline bool isKeyUp(SDL_Keycode key) {
    return isKeyUp(SDL_GetScancodeFromKey(key));
}
```

Se puede quitar la llamada a keyDownEvent, pero hay que estar seguro de que hay un evento correspondiente antes de llamar a isKeyDown porque después de pulsar una tecla, su estado queda igual durante varias iteraciones.

¡Esto no garantiza que la tecla 'key' ha cambiado estado en la última iteración, sólo que su estado actual es UP!

Se puede quitar la llamada a keyUpEvent como en el caso de isKeyDown

La Clase InputHandler: IV

```
inline bool mouseMotionEvent() {
    return isMouseMotionEvent_;
}

inline bool mouseButtonEvent() {
    return isMouseButtonEvent_;
}

inline const std::pair<Sint32, Sint32>& getMousePos() {
    return mousePos_;
}

inline int getMouseButtonState(MOUSEBUTTON b) {
    return mbState_[b];
}
```

Ejercicio

Cambia el InputHandler para que use 2 arrays

```
std::array<bool, SDL_NUM_SCANCODES> keysUp_;
std::array<bool, SDL_NUM_SCANCODES> keysDown_;
```

para el almacenar el estado de la teclas en lugar de usar

```
kbState_ = SDL_GetKeyboardState(0);
```

En este caso, si isKeyUp(key) devuelve true, garantiza que esto ocurrió en la última iteración