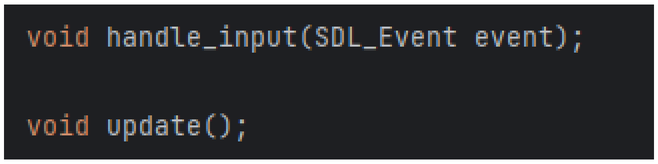
SDL2 Course Day-6
Better Code and Textures

Optimising handle input and update:

Looking at the previous week's code, we will make a change to the *handle input* function of the player object. As a good design practice most code separates out every task into a separate function. To this end we separate the current handle input into two by adding an update function. The current handle input is checking what the input key press is and updating the value of the rectangle's position.

In the Player_Object.h file:



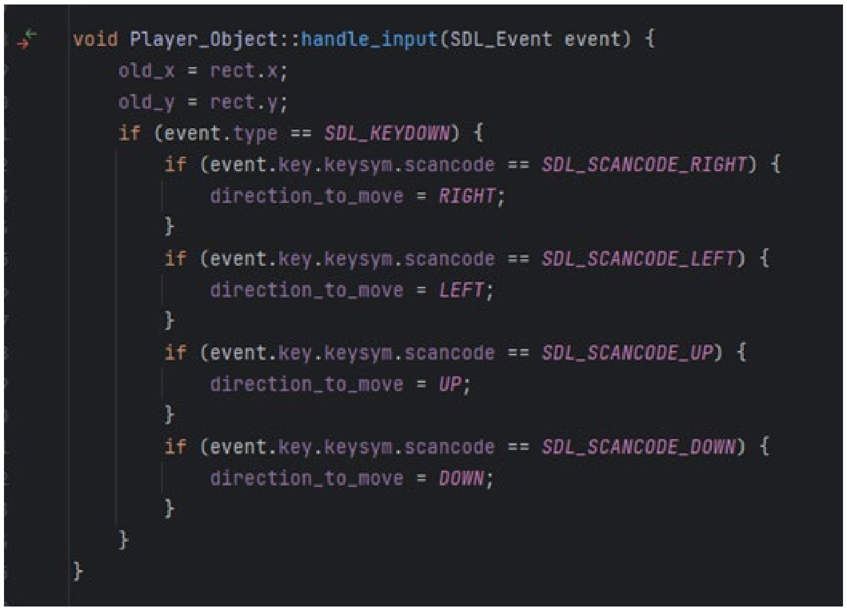
```
void handle_input(SDL_Event event);

void update();
```

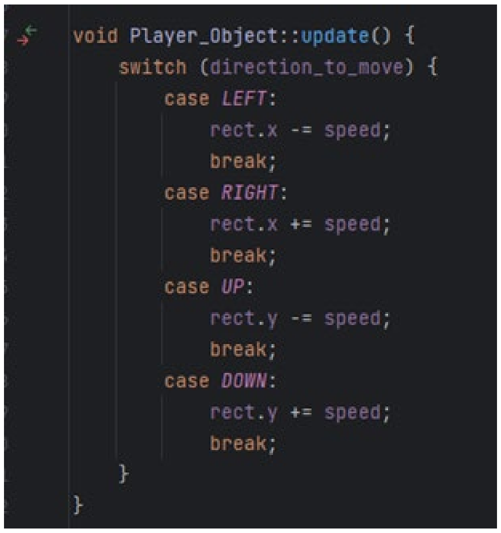
- Add an enum (discussed in next bit) direction property to the player class which stores what direction to move in next.

- Handle Input - check what key is pressed and update the direction.
- Update - changes rectangle coordinates

In Player_Object.cpp:



```
void Player_Object::handle_input(SDL_Event event) {
    old_x = rect.x;
    old_y = rect.y;
    if (event.type == SDL_KEYDOWN) {
        if (event.key.keysym.scancode == SDL_SCANCODE_RIGHT) {
            direction_to_move = RIGHT;
        }
        if (event.key.keysym.scancode == SDL_SCANCODE_LEFT) {
            direction_to_move = LEFT;
        }
        if (event.key.keysym.scancode == SDL_SCANCODE_UP) {
            direction_to_move = UP;
        }
        if (event.key.keysym.scancode == SDL_SCANCODE_DOWN) {
            direction_to_move = DOWN;
        }
    }
}
```



```
void Player_Object::update() {
    switch (direction_to_move) {
        case LEFT:
            rect.x -= speed;
            break;
        case RIGHT:
            rect.x += speed;
            break;
        case UP:
            rect.y -= speed;
            break;
        case DOWN:
            rect.y += speed;
            break;
    }
}
```

Enums:

Enums are structures that can take only one value at a time.
In many web applications you might have seen a drop-down menu. Here you have a list of options and you can choose only one. Similarly, in an enum your variable has multiple options and can have only one of those values at a time.

To create an enum type (enum is a user defined data type- like a struct):

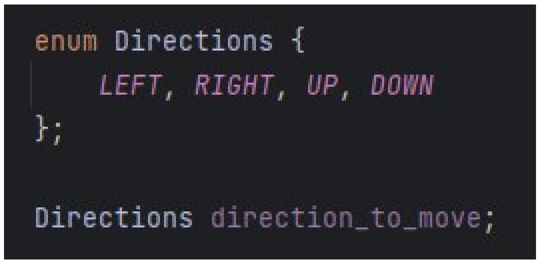
```
enum <enum_name> {
        Value1,
        Value2,
        Value3,
        Value4
}
```

To create an enum variable:

enum_name var_name = {any one of the listed values}

In our program we are using it to assign a direction to each player object- either up, down, left or right. Since it can only be one direction at a time, we use enums.

In .h:



```
enum Directions {
    LEFT, RIGHT, UP, DOWN
};

Directions direction_to_move;
```

Switch:

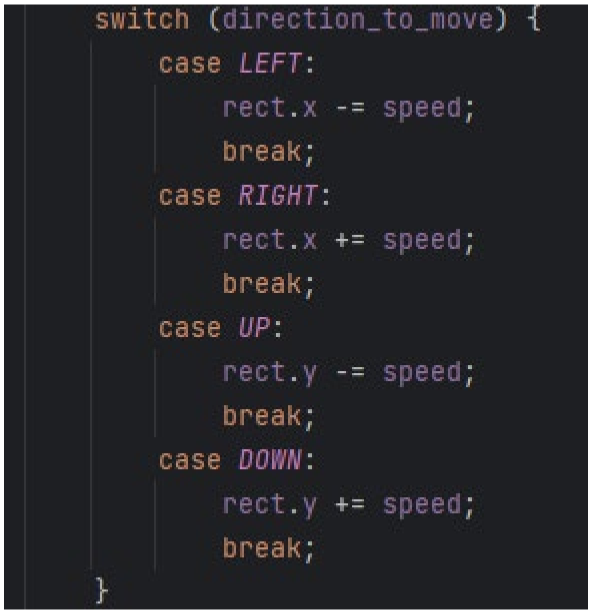
Switch is a modification of an if-else ladder. It executes different parts of code based on the value of an expression. Each value is represented by a case. Use the break statement to tell the compiler where the case code ends. Default case is for all values of the expression not explicitly mentioned in the cases.

```
switch(expression) {
        case value1:
                Statements to execute
                break;

        case value2:
                Statements to execute
                break;

        default:
                Statements to execute
                break;
}
```

We have replaced the if else statement that maps the direction keys to change player position, to a switch.



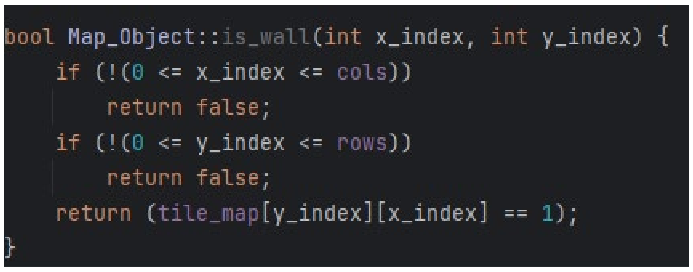
```
switch (direction_to_move) {
    case LEFT:
        rect.x -= speed;
        break;
    case RIGHT:
        rect.x += speed;
        break;
    case UP:
        rect.y -= speed;
        break;
    case DOWN:
        rect.y += speed;
        break;
}
```

Checking map for walls:

Adding an is_wall function to the map object so it can be used for tile by tile movement in the future.

First we check if the value entered is in the scope of the map. If it is, we check if the value of the map at that index is 1 or 0. If 1 we return true else false.

In Map_Object.cpp(make sure you modify the .h file too!):



```
bool Map_Object::is_wall(int x_index, int y_index) {
    if (!(0 <= x_index <= cols))
        return false;
    if (!(0 <= y_index <= rows))
        return false;
    return (tile_map[y_index][x_index] == 1);
}
```

<u>Accessing the map</u>:

To check if the point at the x coordinate and y coordinate is a wall or not how do we check the map?

(Assume that every x and y coordinate of a rectangle aligns exactly with the map tiles. We will learn how to implement movement that facilitates this in the coming classes)

The x coordinate of the rectangle refers to its column (x moves left and right on the tile-map) And y coordinate represents the row (y moves up and down which is represented by different rows of the tile-map)

So we check map[y_coord][x_coord] to check if the map value is 1 or 0.

It's very important to know if x and y coordinates of a point on screen map to the rows or columns on a tile-map. We have made the mistake of inverting them and wondering what is wrong with our code for hours. DON'T MAKE THE SAME MISTAKES! Cool?

OK

Now for the fun stuff-
<u>Sprites and Textures</u>:

Just watching rectangles move on a screen can get pretty boring easily. Let add some images to the screen.

There are 2 kinds of image things (for lack of a better word) in SDL : Surfaces and Textures (represented in SDL by the data-types SDL_Surface and SDL_Texture).

There is some theory behind this, but we felt it a bit too complicated and unnecessary, just use textures :) .

Steps to display an image:

- Create surface
- Load image to surface
- Create texture from the surface
- Delete the surface
- Use the texture everywhere needed
- destroy the texture

Let us display a map texture:

Store the cell width and cell height of a map object first.

```
Map_Object::Map_Object(int cell_w, int cell_h, SDL_Renderer *Renderer)
        : Renderer(Renderer), cell_height(cell_h), cell_width(cell_w) {
```

Load

```
SDL_Surface *dummy = IMG_Load( file: "map.png");
```

Convert surface to texture

```
Map_Texture = SDL_CreateTextureFromSurface(Renderer,  surface: dummy);
```

Destroy surface

```
SDL_FreeSurface( surface: dummy);
```

Destroy texture (we write it in the destructor)

```
Map_Object::~Map_Object() {
    SDL_DestroyTexture( texture: Map_Texture);
}
```

Make sure you modify the .h files as well. Loading the image, converting surface to texture and destroying the surface happens only once in the entire code (written in the constructor) to create a texture. We store it as a property of our Map object to use it later. It is destroyed when the map is deleted (destructor).

<u>Paths/ Directories</u>:

Every program is converted into an executable file in the following steps:
Source code -> compiler -> .exe file

IMP! Do not store your image file in the same directory as your .cpp files. In execution when the image is loaded, it is supposed to be in the directory of the .exe. In our case, the .exe file was specified to be created in the same directory, so we can put it in the same directory:

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR})
```

Focus on the words, runtime_output_directory and current_source_dir; we have set the runtime directory (place where our .exe file will be created) to the same place our current_source_dir is (place where our .cpp files are).

Displaying a texture:

We now have a texture stored in some variable of type SDL_Texture.

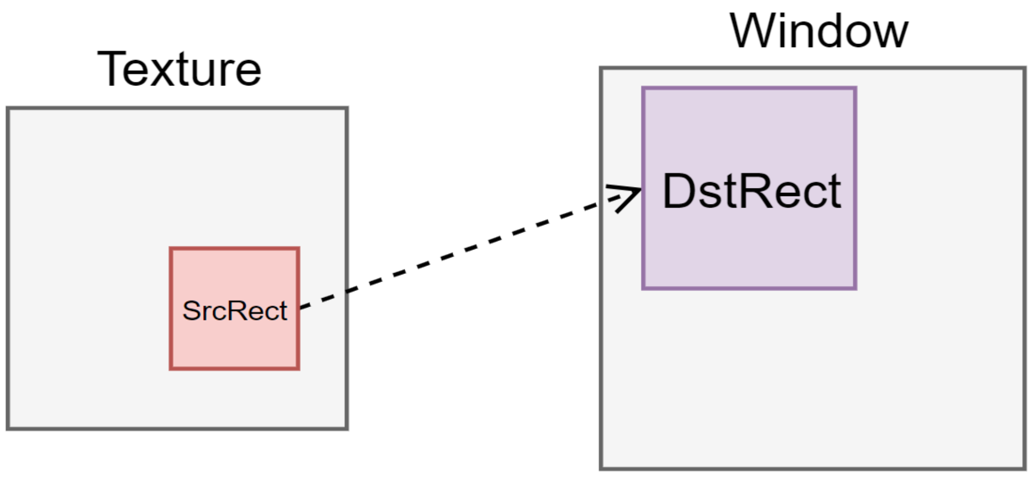
To display a texture, we have the function:

```
SDL_RenderCopy();
```

It takes in the following parameters:

- Renderer – The renderer which we want to use to draw.
- Texture – The texture which we want to draw.
- SrcRect – What part of the texture we want to draw.
  (pass nullptr if you want the entire texture)
- DstRect – Where in the window do we want to draw.
  (pass nullptr if you want to draw it on the entire window).

(If dimensions of SrcRect and DstRect don't match, it gets stretched accordingly.)



Make the following changes to map's draw:

```
void Map_Object::draw() {
//    SDL_SetRenderDrawColor(Renderer, 0, 0, 255, 255);
//    for (auto rect: walls) {
//        SDL_RenderFillRect(Renderer, &rect);
//    }
    SDL_Rect dstRect = {0, 0, cell_width * cols, cell_height * rows};
    SDL_RenderCopy(Renderer, Map_Texture, nullptr, &dstRect);
}
```

Thats all folks!!!

PS: I wrote this at 12 midnight so if its ***wonky*** my apologies(Mandi)

PPS or is it PSS: Please start working on your final projects - groups of 2 max or individual. If you have questions about what to do, how to do, etc? ASK!!!

Bye <3