Platformer levels

Part 1

EXIT
HEALTH
FUEL
BLAST-CARBINE

FPS: 59
TEXTURE: (PRESS N TO SWITCH)
TILED: FALSEPRESS T TO TOGGLE
X: 1828,409 - Y: 1476,546 - ROTATED: 3,150086
COLOR (KEYS UIOP TO CHANGE) --- RED: 1 - GREEN: 1 - BLUE: 1 - ALPHA: 1
MASS: 1PRESS +/- TO INCREASE/DECREASE
MOVEABLE: 0PRESS M TO TOGGLE
NO ROTATE: FALSEPRESS R TO TOGGLE
GEOM FROM TEXTURE: FALSEPRESS Z TO TOGGLE
HURT: 0 - PUSH (A/S): 0
COLLISION TYPE:

Tile-based levels

MARIO
000500
×01
WORLD
1-1
TIME
321

0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
2 3 0 0 0 0 0 0 0 0 0
5 6 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 9 0 0 9 0 0 0 0
0 0 9 9 0 0 9 9 0 0 0
0 9 9 9 0 0 9 9 9 0 0
0 1 2 3 4 5 6 7 8 9 x x x x x x x x x x x x x x

solids = 7,8,9,14,15,20,21,22

1
3
8
16

3D
OFF

1
4
4
$0

Platformer Demo

Defining a tile map level

0
1
2
3
4
5
6
7
8
9
10
11
12
...

Storing level data as a 2-dimensional array

```
unsigned int levelData[LEVEL_HEIGHT][LEVEL_WIDTH];
```

```
{0,20,4,4,4,4,4,4,0,0,0,0,0,0,4,4,4,4,4,4,20,0};
```

```
#define LEVEL_HEIGHT 16
#define LEVEL_WIDTH 22

unsigned int levelData[LEVEL_HEIGHT][LEVEL_WIDTH] =
{
    {11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11},
    {0,20,4,4,4,4,4,4,0,0,0,0,0,0,4,4,4,4,4,4,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,6,6,6,6,6,6,6,6,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,6,6,6,6,6,0,0,0,0,0,0,0,0,6,6,6,6,6,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,6,6,6,6,6,6,6,6,0,0,0,0,0,20,0},
    {0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,0},
    {0,20,125,118,0,0,116,0,0,0,0,0,0,0,0,0,0,117,0,127,20,0},
    {2,2,2,2,2,2,2,2,2,3,0,0,1,2,2,2,2,2,2,2,2,2},
    {32,33,33,34,32,33,33,34,33,35,100,101,35,32,33,32,34,32,33,32,33,33}
};
```

Rendering a tile map level

Need to render the entire tilemap as one vertex array

Hello

Go through the tiles **line by line** and **add vertices to a single array**.

(Keep in mind that our **Y axis** points **upwards**, while tile **indexes count downwards**)

```
std::vector<float> vertexData;
std::vector<float> texCoordData;

for(int y=0; y < LEVEL_HEIGHT; y++) {
    for(int x=0; x < LEVEL_WIDTH; x++) {

            float u = (float)(((int)levelData[y][x]) % SPRITE_COUNT_X) / (float) SPRITE_COUNT_X;
            float v = (float)(((int)levelData[y][x]) / SPRITE_COUNT_X) / (float) SPRITE_COUNT_Y;

            float spriteWidth = 1.0f/(float)SPRITE_COUNT_X;
            float spriteHeight = 1.0f/(float)SPRITE_COUNT_Y;

            vertexData.insert(vertexData.end(), {

                TILE_SIZE * x, -TILE_SIZE * y,
                TILE_SIZE * x, (-TILE_SIZE * y)-TILE_SIZE,
                (TILE_SIZE * x)+TILE_SIZE, (-TILE_SIZE * y)-TILE_SIZE,

                TILE_SIZE * x, -TILE_SIZE * y,
                (TILE_SIZE * x)+TILE_SIZE, (-TILE_SIZE * y)-TILE_SIZE,
                (TILE_SIZE * x)+TILE_SIZE, -TILE_SIZE * y
            });

            texCoordData.insert(texCoordData.end(), {
                u, v,
                u, v+(spriteHeight),
                u+spriteWidth, v+(spriteHeight),

                u, v,
                u+spriteWidth, v+(spriteHeight),
                u+spriteWidth, v
            });
    }
}
```

We can optimize by not drawing empty grid tiles at all

```
for(int y=0; y < LEVEL_HEIGHT; y++) {
    for(int x=0; x < LEVEL_WIDTH; x++) {

        if(levelData[y][x] != 0) {
             // add vertices
        }
    }
}
```

Colliding with a tilemap

Separate axis movement again!

First the Y axis…

Convert our entity position
to the grid coordinates and check
if that tile is solid.

Divide your unit coordinates by the tile size and invert Y.

```
void worldToTileCoordinates(float worldX, float worldY, int *gridX, int *gridY) {
    *gridX = (int)(worldX / TILE_SIZE);
    *gridY = (int)(worldY / -TILE_SIZE);
}
```

What is a solid tile?

It’s up to you!
Define the tile indexes that are considered solid!

Make sure that your tile coordinates
are not negative or larger than your map!

We don’t want to collide using the entity’s center, so check the point at the bottom (-half height).

-TILE_SIZE * row
entity.y - entity.height/2

If the tile is solid, adjust our Y-coordinate up by the difference between the point we are testing and the top of the tile (plus a tiny amount).

Don’t forget to reset velocity to 0 and set your collision flags!

Now, check the point at the top…

entity.y + entity.height/2
(-TILE_SIZE * tile_y) - TILE_SIZE

If the tile is solid, adjust our Y-coordinate down by the difference between the point we are testing and the bottom of the tile (plus a tiny amount).

Now the X axis…

On the left…

entity.x - entity.width/2
(TILE_SIZE * tile_x) + TILE_SIZE

If the tile is solid, adjust our X-coordinate right by the difference between the point we are testing and the right of the tile (plus a tiny amount).

Now finally on the right…

(TILE_SIZE * tile_x)
entity.x + entity.width/2

If the tile is solid, adjust our X-coordinate left by the difference between the point we are testing and the left of the tile (plus a tiny amount).

You may test additional points along the sprite if it is larger than a tile.

Scrolling

What is scrolling?

We need to translate everything so that the player is in the center of the screen.

We need to translate everything so that the player is in the center of the screen.

The view matrix!

Translate the view matrix by the inverse of the player's position!
(or whatever you want to center on)!