Hadí jazyková sbírka (Snakey language collection)

Ladislav Dobrovský
ladislav.dobrovsky@gmail.com

(works at CEITEC BUT with HPC and embedded stuff)

SnakeGameCollection (hobby project)

- github.com/karna48/SnakeGameCollection
- Python:
  - Pyglet (OpenGL inside)
  - Pyodide + HTML canvas (no WebGL)
- C++: SDL2 + OpenGL
  - Deprecated intermediate mode
  - Vertex Buffer Object + GLSL shaders
  - Geometry shader
- C#, Java, Julia, Matlab, Octave, Vala

LIVE DEMO!

Python + Pyglet

```
Rabbit = namedtuple('Rabbit', ('row', 'col', 'sprite'))
SnakePart = namedtuple('SnakePart', ('row', 'col', 'dir', 'sprite'))

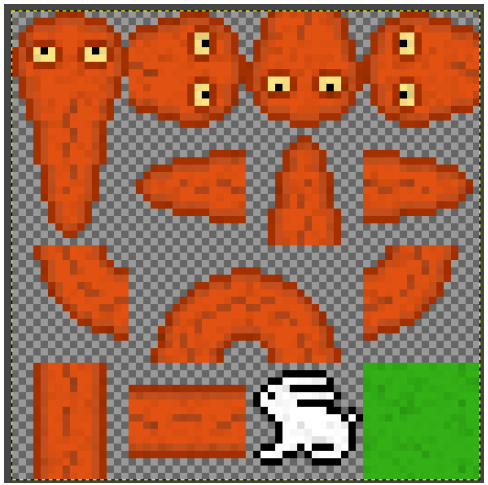
class MySprite(pyglet.sprite.Sprite): ...  # handles scale internaly!

class SnakeGame:
  def __init__(self):
        self.main_window = pyglet.window.Window(*WINDOW_SIZE)
        self.main_window.push_handlers(self)
        pyglet.clock.schedule_interval(self.update, 1/60)
        ...
    def on_draw(self):
    def on_key_press(self, symbol, _modifiers):
    def on_mouse_press(self, x, y, b, _mod): …
    def update(self, dt): ...

    def reset_snake(self): ...
    def place_rabbit(self): ...

game_state = SnakeGame()
pyglet.app.run()
```

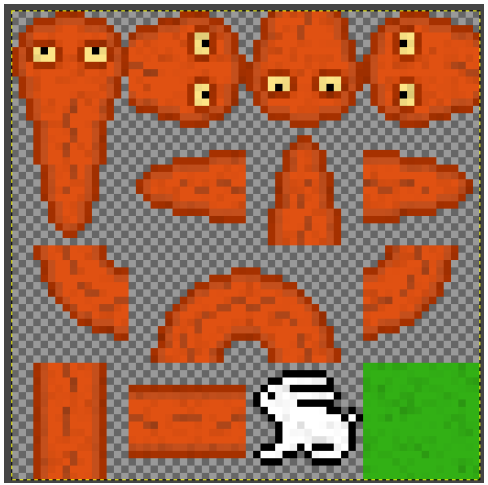
Carving up the image - Pyglet



```
self.images = {
    name: snake_png.get_region(
        j * self.sprite_w, (3-i) * self.sprite_h,
        self.sprite_w, self.sprite_h
    )
    for i, names_row in enumerate(
        (('head_up', 'head_right', 'head_down', 'head_left'),
         ('tail_up', 'tail_right', 'tail_down', 'tail_left'),
         ('turn_1', 'turn_2', 'turn_3', 'turn_4'),
         ('vertical', 'horizontal', 'rabbit', 'grass'))
    )
    for j, name in enumerate(names_row)
}
```

```
MySprite(self.images[name], ...)
```

Carving up the image - OpenGL



```
const std::vector<std::vector<std::string>> IMG_NAMES{
    {"head_up", "head_right", "head_down", "head_left"},
    {"tail_up", "tail_right", "tail_down", "tail_left"},
    {"turn_1", "turn_2", "turn_3", "turn_4"},
    {"vertical", "horizontal", "rabbit", "grass"}
};
const auto g_img_rc = [](){
    std::unordered_map<std::string, std::pair<int, int>> img_rc;
    int i = 0;
    for(auto const& names_row : IMG_NAMES)
    {
        int j = 0;
        for(auto const& name : names_row) {
            img_rc[name] = std::pair<int, int>(i, j);
            j++;
        }
        i++;
    }
    return img_rc;
}();

void Sprite::set_image(const std::string& img_name)
{
    auto [i, j] = g_img_rc.at(img_name);
    u1 = 0.25f * j;         v1 = 0.25f * (i+1);
    u2 = 0.25f * (j+1);     v2 = 0.25f * i;
}
```

Input – requests for action

```
def on_key_press(self, symbol, _modifiers):
    if symbol == key.LEFT:
        if head_dir != 'right':
            self.snake_dir_next = 'left'
    elif symbol == key.RIGHT:
        if head_dir != 'left':
            self.snake_dir_next = 'right'
    elif symbol == key.UP:
        if head_dir != 'down':
            self.snake_dir_next = 'up'
    elif symbol == key.DOWN:
        if head_dir != 'up':
            self.snake_dir_next = 'down'
```

Changing state with time

```
def update(self, dt):
    self.snake_move_t_rem -= dt
    if self.snake_move_t_rem <= 0:
        self.snake_move_t_rem = self.snake_move_t
        head_dir = self.snake_dir_next
        row = self.snake[0].row
        col = self.snake[0].col
        if head_dir == 'left':
            col -= 1
        elif head_dir == 'right':
            col += 1
        elif head_dir == 'up':
            row += 1
        elif head_dir == 'down':
            row -= 1
        row %= self.rows
        col %= self.columns

        self.snake.insert(0,  SnakePart(row, col, head_dir, MySprite(…)))
        # (… handle graphics update...)
        rabbit_eaten = row == self.rabbit.row and col == self.rabbit.col

        if not rabbit_eaten:
            self.snake.pop()   # del self.snake[-1]
            tail = self.snake[-1]
            tail.sprite.image = self.images['tail_'+self.snake[-2].dir]
```

Finding free square

```
self.set_all_squares = set(product(range(self.rows), range(self.columns)))
...
set_snake_squares = set((s[0], s[1]) for s in self.snake)
square = random.choice(list(self.set_all_squares-set_snake_squares))
```

```
auto it=std::set_difference(
   set_all_squares.begin(), set_all_squares.end(),
   set_snake_squares.begin(), set_snake_squares.end(),
   free_squares.begin());
free_squares.resize(it-free_squares.begin());

int i = std::uniform_int_distribution<int>(0, free_squares.size()-1)(rnd_generator);
auto [row, col] = free_squares[i];
```

```
B = zeros(length(app.parts), 2);
for i = 1:length(app.parts)
    B(i, :) = [app.parts(i).row, app.parts(i).col];
end
S = setdiff(app.A_set, B, 'rows');
i = randi(size(S, 1));
app.rabbit.row = S(i, 1);    app.rabbit.col = S(i, 2);
```

Geometry shader

```
#version 330 core
layout (points) in;
layout (triangle_strip, max_vertices = 4) out;
out vec2 UV;
uniform mat4 projection;
uniform float sprite_size;

void main() {
    vec4 p = vec4(gl_in[0].gl_Position.xy, 0.0f, 1.0f);
    int img_linidx = int(gl_in[0].gl_Position.z);
    int img_row = img_linidx / 4;
    int img_col = img_linidx - img_row*4;
    vec2 t = vec2(float(img_col), float(img_row)) * 0.25f;

    gl_Position = projection * p;
    UV = t+vec2(0.0f, 0.25f);
    EmitVertex();
    gl_Position = projection * (p+vec4(sprite_size, 0.0f, 0.0f, 0.0f));
    UV = t+vec2(0.25f, 0.25f);
    EmitVertex();
    gl_Position = projection * (p+vec4(0.0f, sprite_size, 0.0f, 0.0f));
    UV = t+vec2(0.0f, 0.0f);
    EmitVertex();
    gl_Position = projection * (p+vec4(sprite_size, sprite_size, 0.0f, 0.0f));
    UV = t+vec2(0.25f, 0.0f);
    EmitVertex();
}
```

Thanks for the attention.

Questions?