Project SEAM

A language

Akira Baruah[*1], Maclyn Brandwein[†1], Sean Inouye[‡1], and Edmund Qiu[§1]

[1]COMS 4115, Columbia University

September 30, 2015

1 Purpose

Many programs, from games to simulations, operate using many discrete objects that all recalculate their own state a fixed number of times per second. Most existing languages provide no built in keywords or syntax for entity management, and those that come closest do so through the abstraction of objects. We propose a language with a simple syntax that makes it easy to create, destroy, and run many simulated entities to ease the work of constructing simulations.

2 Overview

The language functions by compiling SEAM's syntax to an intermediate low-level language which, when combined with some intermediary boilerplate code, allows the rapid prototyping of complex simulations. We also provide simple optional drawing and input functions which are attached by the language to entities. For example, when writing a Breakout clone, each block is represented by an entity which awaits collision results, the ball is its own entity which moves on every step, and the paddle entity exists to draw the paddle and receive key events.

3 Features

- Concept of contained elements called *entities* with a step function called n (user-configurable value) times per minute
- Special member functions for the entities that are invoked by the system when relevant (`draw()`, when drawing is to occur, `key()`, when a user hits a key)
- Management of when to called `step`, `draw`, and `key` functions for entities
- Easy creation of entity with `spawn_entity(...)` and `kill_entity(...)`
- Proper scope access from entity

[*] akb2158
[†] mgb2163
[‡] si2281
[§] ejq2106

4 Examples

Simple breakout game:

```
boolean ball_released = false
boolean release_ball_flag = false

int block_width = 10
int block_height = 4

object ball_object
object paddle_object

function abs(float val) {
    if (val < 0.0) {
        return -1.0 * val
    } else {
        return val
    }
};

function aabb_collision(float ax, float ay, float awidth, float aheight, float bx, \
    float by, float bwidth, float bheight)
{
    return (abs(ax - bx) * 2 < (awidth + bwidth)) (abs(ay - by) * 2 < (aheight +
};

function main {
    // Spawn the blocks
    for (int row = 0, row < 4, row += 1) {
        for (int col = 0, col < 10, col += 1) {
            spawn_entity(block, block_startx + col * block_width, \
block_starty + row * block_height)
        }
    }

    // Spawn player and ball:
    paddle_object = spawn_controlled_entity(paddle, 150, 0)
    ball_object = spawn_entity(ball, 150, 20)
};

entity ball {
    bounce(horizontal_bounce) {
        if (horizontal_bounce) {
            vx = -vx
        } else {
            // vertical bounce
            vy = -vy
        }
    };
    step {
        if (ball_released) {
            // Simplified - should actually bounce off of walls and
            // if it goes out of bounds, then it will set ball_released to
            // false again
            x += vx * (1/60)  // timestep
            y += vy * (1/60)
        } else {
            // Whenever the ball isn't moving around, just
            // follow the paddle around
            vx = 0
            vy = 0
            x = paddle.x
            y = paddle.y

            // Once it is released, then launch the ball
            if (release_ball_flag) {
                vx = -5
                vy = 10
                release_ball_flag = false
                ball_released = true
            }
        }
    };
};

entity paddle {
    key(int key) {
        // Dummy key codes, but will define real ones later
        if (key == 0) {
            x -= 5
        } elseif (key == 1) {
            x += 5
        } elseif (key == 2) {
            // And here we go
            release_ball_flag = true
        }
    };
};

entity block {
    step {
        // Something like this, if it's of size 5 x 5
        boolean result = aabb_collision(x, y, block_width, block_height, \
            ball_object.x, ball_object.y, ball_object.width, ball_object.height)

        // Collision!
        if (result) {
            entitydeath // Marks death of this entity
            boolean horizontal_bounce = ... // math for which direction of contact
            ball.bounce(horizontal_bounce) // ball now reacts accordingly
        }
    };
```

```
};
```

5 Milestones

Fundamental compiler components (weeks 5 - 9)

- Finalize syntax
- Implement basic lexical analyzer
- Implement basic semantic analyzer

Questions:

- How is the compiler going to break down the incoming stream of data?
- What type of algorithms and mathematical models will be implemented?
- What type of data types will the language accept

First Working Implementation (weeks 10 - 12)

- Implement debug input/output
- Implement "Hello World"
- Implement early graphics

Finalize deliverables (weeks 13-15)

- Finalize all tests
- Write code examples
- Iterate test and debugging

6 Team Roles

Role	Name	UNI
Manager	Sean	`si2281`
Language Guru	Maclyn	`mgb2163`
System Architect	Akira	`akb2158`
Tester	Edmund	`ejq2106`