Lucifer

PLT Final Presentation

- Manager - Michael Fagan
- Lang Guru - Elliott Morelli
- Architect - Cherry Chu
- Tester - Robert Becker

Introduction

Introduction

- Motivation
  - Game development
    - Typical roadblocks
    - Cross-platform
- Idea
  - SDL integration
    - Why SDL?
    - SDL Uses & benefits
  - XServer
  - 2D games

Feature Highlights

Built-In Objects

- Entity

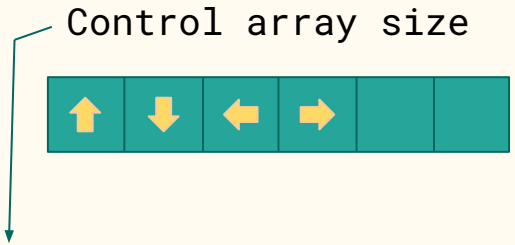
```
Entity e;

e = new Entity(0, 0, “rock.png”);
```

- Player

```
Player p;

p = new Player(100, 100, “bird.png”, 6);
```



Function Usage

- Declaration

```
fun add int (int a, int b) { return a + b; }
```

- Global function call

```
add(1, 2);
```

- Object function call (Entity e)

```
e.changeEntityX(-10);
```

```
changeEntityX(e, -10);
```

Rungame Loop

```
e = new Entity(100, 100, “spaceship.png”);

x = 600;

runGame(x > 0; 1) {

  x = x - 1;

  e.changeEntityX(-10);

}
```

SDL & Architecture

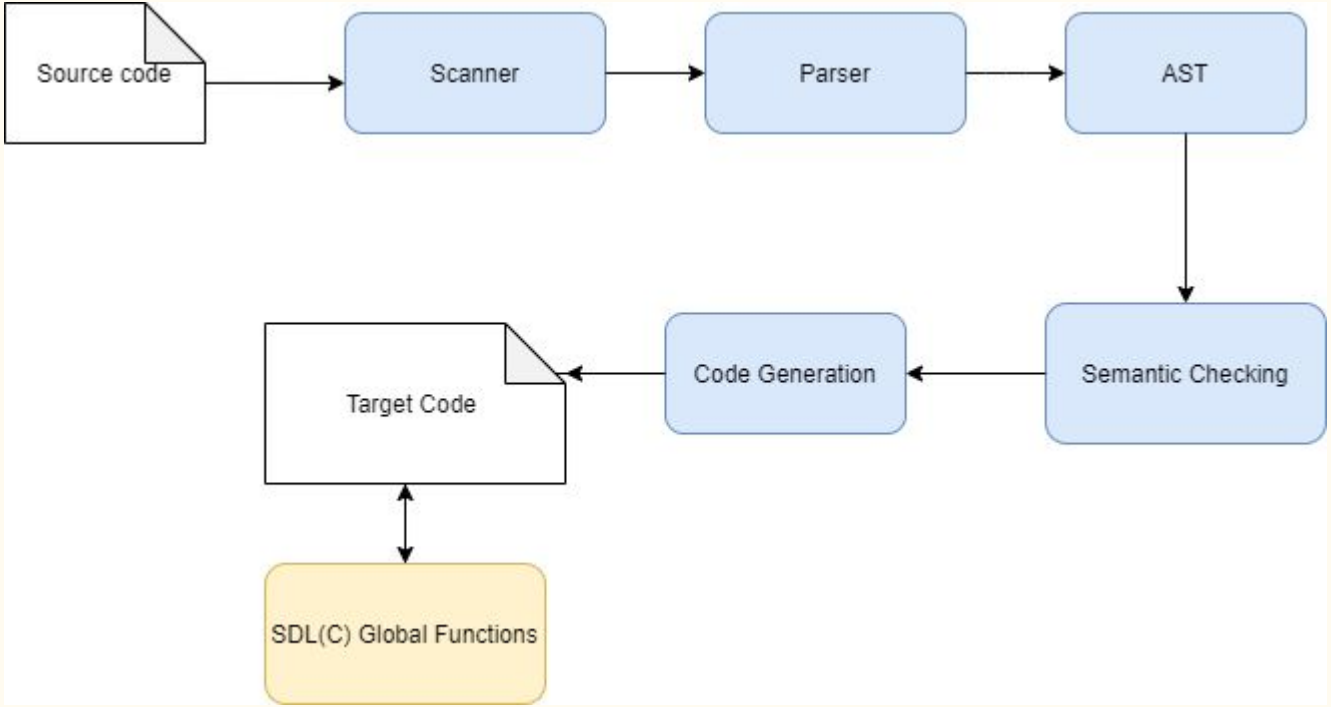
SDL

- Codegen enlists global C functions to interact with the SDL library
  - C functions prepareScene() and presentScene() use SDL_SetRenderClear();
  - Uses C struct “app” which stores SDL_Renderer, SDL_Window, int[] keyboard
- runGame() is responsible for rendering Entities and Players
  - Calls to:
  - SDL_LoadTexture()
  - SDL_QueryTexture()
  - SDL_RenderCopy()
- Lucifer links with SDL(C) at runtime

SDL

- Keyboard input
    - Global fn: isKeyPressed(int key);
    - Player controls array: p.controlPlayer(int speed);
- Running a Lucifer .exe that initializes SDL
    - A graphics window will open
    - Any rendered textures will be visible at their initial positions on SDL's (x,y) plane

Architecture



Testing

Testing

- Test suite is comprised of Pass (“test”) cases, Fail cases, and Visual cases
- We have over 90 non-visual tests to run, so these are all checked by our shell script
  - Each pass case’s output is checked against a .out file, while each Fail case’s error message is tested against a .err file
  - If a test’s result doesn’t match its corresponding file, the script prints a statement describing how it failed, and saves the unexpected output and its difference from the expected output in the current directory
  - A more detailed report of test results are saved in the testall.log file that is generated afterwards
- Visual cases involve rendering a scene where the user can control what is displayed, so these cases cannot reliably be fully tested through the script
  - Instead, the shell script creates an executable file for each test, which can then be run on a machine with an active X Server

Code Demo

```
fun checkCollision bool (Entity e, Player p){
    int pRightX;
    int pDownY;
    int eRightX;
    int eDownY;

    pRightX = p.getPlayerX() + p.getPlayerHx();
    pDownY = p.getPlayerY() + p.getPlayerHy();
    eRightX = e.getEntityX() + e.getEntityHx();
    eDownY = e.getEntityY() + e.getEntityHy();

    if(p.getPlayerX() == pRightX || p.getPlayerY() == pDownY || e.getEntityX() ==
eRightX || e.getEntityY() == eDownY){
        return false;
    }

    if(p.getPlayerX() >= eRightX || e.getEntityX() >= pRightX ){
        return false;
    }

    if(p.getPlayerY() >= eDownY || e.getEntityY() >= pDownY){
        return false;
    }
    return true;
}
```

```
fun main void () {

Player knight;
Entity e;
Entity e2;
Entity e3;

Entity winscreen;
running = true;
… /*more globals*/

 runGame(running; 60){

        knight.addPlayerHitBox(80, 80);
        e.addEntityHitBox(100, 100);
        e2.addEntityHitBox(120, 120);
        e3.addEntityHitBox(120, 120);

        collide = (checkCollision(e,knight) || checkCollision(e2,knight) ||
checkCollision(e3,knight));

        knight.controlPlayer(15);
/*...more loop logic… but time for demo!*/
```

Thank you