自作言語
Naughtiness

@Sin0n0me

自己紹介

- インターネット上ではSin0n0meとして活動中
  - Twitter: @Sin0n0meSan2
- SecHack365 ‘24 川合ゼミ 修了

イラストは学校の友人に
描いて頂いた(可愛い)
ガルト。さん
@NamikimithiS

趣味は

- プログラミング
- ゲーム
- 旅
- 歩くこと

自作言語のNaughtinessって?

SecHack365で作成した自作言語(省略してナギと呼んでる)

バックエンド部分はLLVMに全任せでフロントエンド部分だけ自作

SecHack365での自作言語のコンセプトは

「ゲーム開発者がチート対策を意識しなくていい言語」

何故自作言語? ライブラリじゃダメ?

- ライブラリやマクロだとコード上で目に見える=意識してしまう
  - 言語側で対策そのものを隠すことで実現できると思った
    - この隠すが子供のいたずらみたいと思ったので Naughtinessという名前になった
- 言語特有の書き方ができる
  - ゲーム特化な書き方ができると思った

何故自作言語? ライブラリじゃダメ?

- ライブラリやマクロだとコード上で目に見える=意識してしまう
  - 言語側で対策そのものを隠すことで実現できると思った
    - この隠すが子供のいたずらみたいと思ったので Naughtinessという名前になった
- 言語特有の書き方ができる
  - ゲーム特化な書き方ができると思った

なにより…

言語を作りたかったから (重要)

具体的な仕組み

意味解析後のAST(抽象構文木)にチート対策の構文木を付け足している

例えば実行時のメモリ内の値の検索を妨げるために簡単なxor演算で使用直前まで見えなくする場合

X = 10;

Y = X;

のような式なら

X = 10 ^ 12648430

Y = X ^ 12648430

になる

構文

基本的に構文はRustライクな構文

特に理由はなく

Rustライクな理由はRustで書いてるから

機能

- 変数宣言
- 条件分岐
- 繰り返し
- 関数呼び出し
- 関数定義
- 構造体定義
- 配列の作成

```
struct Nagi {
    hoge: i32,
    fuga: u32,
    wan: i64,
    nya: u64,
}

fn main() {
    // this is comment
    let hoge = 100;
    let fuga = [0, 1, 2];

    if hoge == 100 {
        sub();
    } else {
        function(1, 20);
    }
}

fn sub() {
    let hoge = (1 + 2) * (3 + 8) * (10 + 32);
}

fn function(a: i32, b: i32) {

}
```

ここまではSecHack365の話
ここからはラボユースでやっていること

ラボユースでやっていること

SecHack365での自作言語の続き

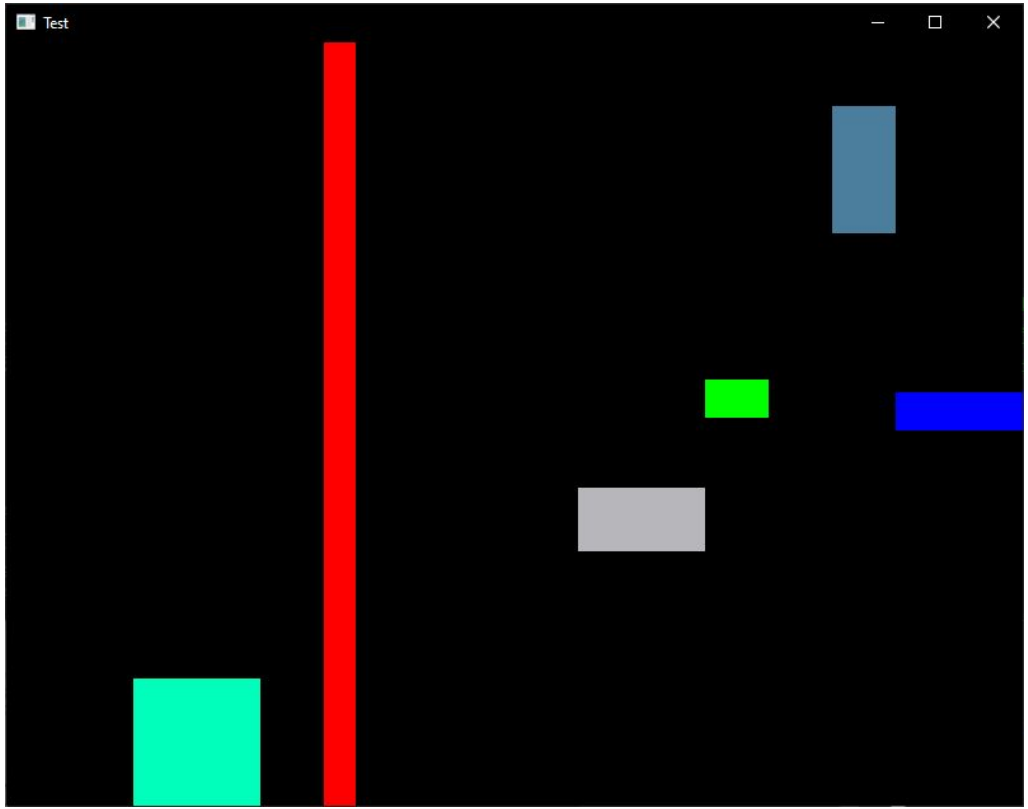
SecHack365では四則演算, 論理演算, 条件分岐などの基本的なことは出来たものの関数呼び出しや構造体定義などができなかったのでその実装

ゲーム開発者向けの言語らしくウィンドウ処理やグラフィカルな処理ができるような実装

合宿中にやったこと

なんとか描画はできた

内部的にはRustで作った静的ライブラリを自作言語側で呼び出してるだけ



合宿後の改善や実装予定

- 再帰下降で構文解析を行っているのでネストが深いとスタックオーバーフローが起きて解析出来ない
  - 再帰下降で解析したいのでスタックを使って対処予定
- 範囲付きの変数の概念
  - 範囲付きの変数を使用すると内部では使用前に変数の範囲チェックがされる
- 3Dモデルの読み込み