ECSInject

A Programming Language built for Entity Component Systems

Jan Kleinmann, Marius Braun

12/2/2025

Agenda

- Motivation
- Entity Component System (ECS)
- Bevy
- ECSinject and what we want to accomplish
- Grammar and AST
- Interpreter
- What is yet to come

Motivation

- Project using a game engine with an Entity Component System (ECS)
- First task in this module: Develop a DSL for a game developed in an ECS (Dungeon)
- PROBLEMS
- Thankfully, the task was changed to work on ANY topic related to programming languages

Entity Component System (ECS)

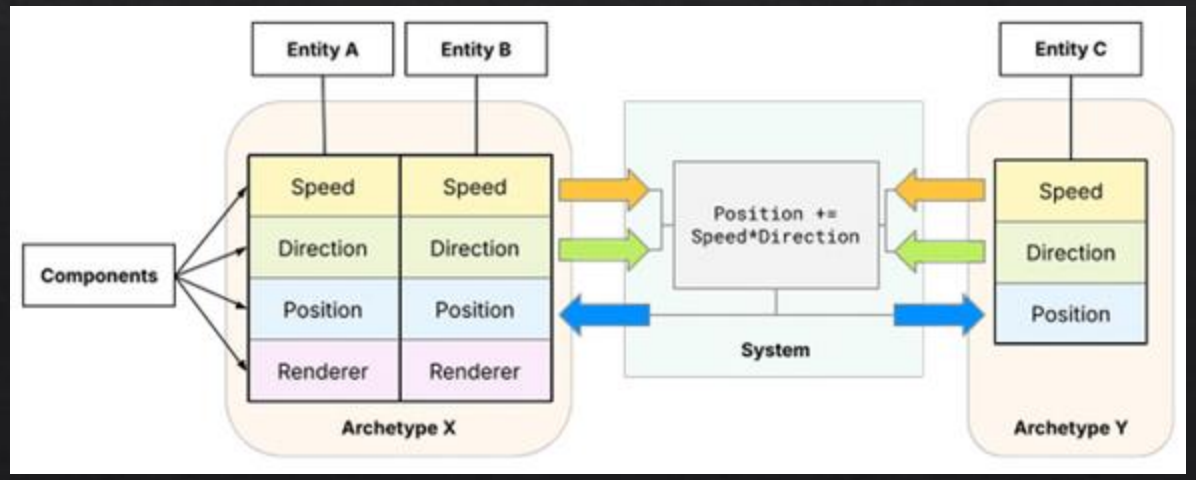
- Software architectural pattern often used in video games
- Composition over inheritance
- Separate data and behaviour
- Enables dynamic and modular game development

Entity Component System (ECS)

- Entity
  - An object
  - Just an ID
- Component
  - Describes attributes/features of Entity
  - Stores Data
- System
  - Behavior associated with attributes
  - Acts on entities with certain components
- World
  - Collection of entities

Entity Component System (ECS)

- A Player is an Entity
- A Player has components
  - Speed
  - Direction
  - Position
- A system acts on the components to move the Player



https://docs.unity3d.com/Packages/com.unity.entities@6.4/manual/concepts-entities.html

Advantages

- Reusability of components
- Simple, modular code
- Integration of new features
- Data and functionality separation

Disadvantages

- Limited Recognition
- Risk of inefficient code
- Build around a language
  - Missing language ergonomics

ECS in Practice

Systems in the Dungeon

```java
public class FogOfWarSystem extends System {
  private static final int DISTANCE_TRANSITION_SIZE = 2; // size of distance transition (in tiles)
  private static final int HIDE_ENTITY_THRESHOLD =
      0xFFFFFF99; // tint color threshold for hiding entities
  private static final int[][] mult = { // needed for casting light
    {1, 0, 0, -1}, {0, 1, -1, 0}, {0, -1, -1, 0}, {-1, 0, 0, -1},
    {-1, 0, 0, 1}, {0, -1, 1, 0}, {0, 1, 1, 0}, {1, 0, 0, 1}
  };
  private static final float TINT_COLOR_WALL_DISTANCE_SCALE =
      1.5f; // scale factor for behind wall distance fog
  private static final float TINT_COLOR_DISTANCE_SCALE = .5f; // scale factor for distance fog

  /** The view distance (range for tiles that are fully visible). */
  private static int currentViewDistance = 7;

  /** The maximum view distance (all tiles to consider for calculation). */
  private static final int MAX_VIEW_DISTANCE = 25;

  private final Map<Tile, Integer> darkenedTiles = new HashMap<>();
  private final List<Entity> hiddenEntities = new ArrayList<>();
  private boolean active = true;


  public void execute() {
    if (!active) return;

    Point heroPos = EntityUtils.getHeroPosition();
    if (heroPos == null) return; // no hero, no fog of war

    List<Tile> allTilesInView = LevelUtils.tilesInRange(heroPos, MAX_VIEW_DISTANCE);
    // Revert all darkened tiles back to light that are not in view
    List<Tile> tilesOutsideView = new ArrayList<>(darkenedTiles.keySet());
    tilesOutsideView.removeAll(allTilesInView);
    revertTilesBackToLight(tilesOutsideView);
   [...]
}
```

Query in the Dungeon

- Cannot specify which components are not needed / must be excluded
- Fetch Components that are needed for calculations

```
1  Map<Boolean, List<HSData>> deadOrAlive =
2      filteredEntityStream(HealthComponent.class, DrawComponent.class)
3          .map(
4              e ->
5                  new HSData(
6                      e,
7                      e.fetch(HealthComponent.class).orElseThrow(),
8                      e.fetch(DrawComponent.class).orElseThrow()))
9          .collect(Collectors.partitioningBy(hsd -> hsd.hc().isDead()));
10
```

Scheduling of Systems in the Dungeon?

- Sometimes needed because of assumed order of operations
  - Systems A does something, that System B requires, but should not wait one frame to execute

No System scheduling!

Bevy game engine

- Systems directly query based on conditions
- Allows for simple parallelism + System scheduling
- Better ergonomics

Bevy – Examples

```rust
fn update_people(mut query: Query<&mut Name, With<Person>>) {
    for mut name in &mut query {
        if name.0 == "Elaina Proctor" {
            name.0 = "Elaina Hume".to_string();
            break; // We don't need to change any other names.
        }
    }
}

```

Example from https://bevy.org/learn/quick-start/getting-started/ecs/

Bevy – Examples

```
fn check_intersections(
    mut commands: Commands,
    pickable: Query<
        (
            &GlobalTransform,
            Entity,
            [...]
        ),
        (Without<GripLeft>, Without<GripRight>),
    >,
    left_tracked: Single<(&GlobalTransform, Entity), With<GripLeft>>,
    right_tracked: Single<(&GlobalTransform, Entity), With<GripRight>>,
    [...]
) {
```

Bevy vs Dungeon

```rust
fn update_people(mut query: Query<&mut Name, With<Person>>) {
    for mut name in &mut query {
        if name.0 == "Elaina Proctor" {
            name.0 = "Elaina Hume".to_string();
            break; // We don't need to change any other names.
        }
    }
}

```

```
Map<Boolean, List<HSData>> deadOrAlive =
    filteredEntityStream(HealthComponent.class, DrawComponent.class
        .map(
            e ->
                new HSData(
                    e,
                    e.fetch(HealthComponent.class).orElseThrow(),
                    e.fetch(DrawComponent.class).orElseThrow()))
        .collect(Collectors.partitioningBy(hsd -> hsd.hc().isDead()
```

Example from https://bevy.org/learn/quick-start/getting-started/ecs/

Bevy – System Scheduling

```
app.add_systems(
    PreUpdate,
    sync_actions
        .before(OxrActionSetSyncSet)
        .run_if(openxr_session_running),
);

app.add_systems(PostUpdate, handle_enable_ortho_camera);
app.add_systems(
    PostUpdate,
    update_ortho_camera_positions.before(handle_enable_ortho_camera),
);
```

The goal?

- Bevy is well designed and works great
  - Slow compile times (macro usage and code generation)
  - But still build around Rust (lifetime and borrow checker restrictions)

- Goal:
  - Design a language for an ECS, that abstracts away all implementation details of the ECS
  - ECS designed for language ⇔ Language designed for ECS
  - Strong query functionality with support for parallel system execution

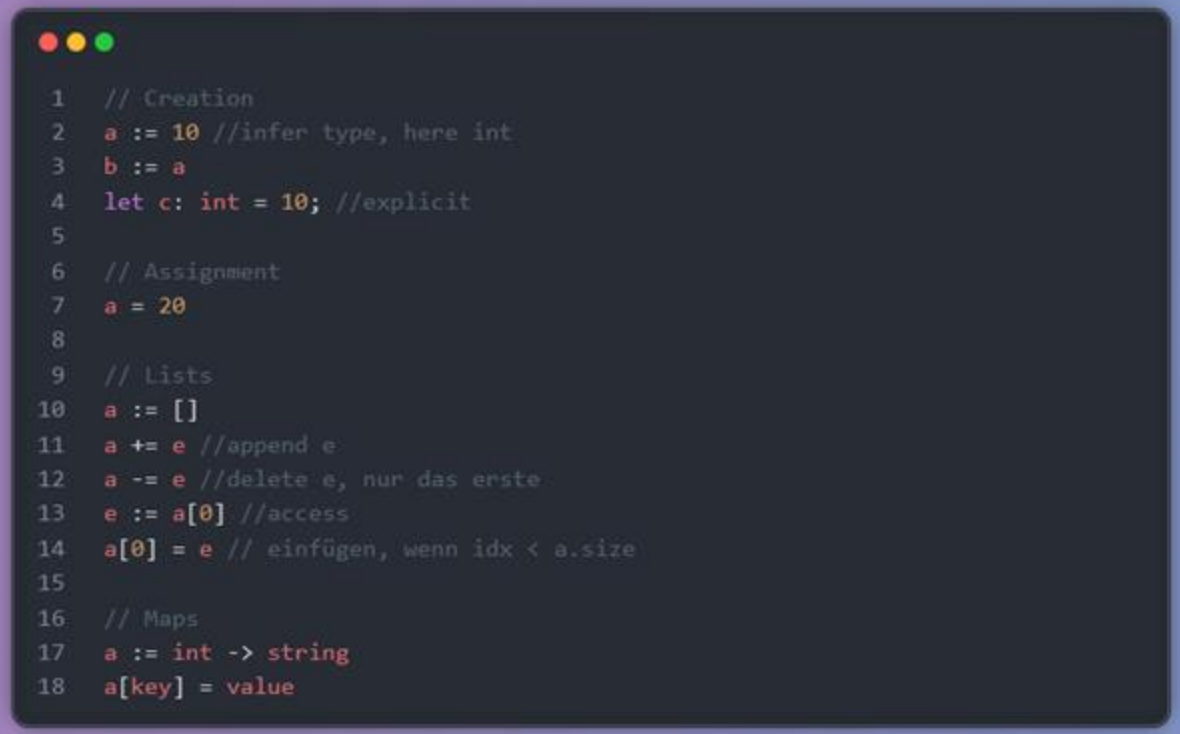
Concept

Concept (Marius)

- Programming language with Entities, Components and Systems as first-class citizens
  - Usable with an ECS
  - Query functionality for systems
- Strong typing
- Type Inference
- No inheritance
- Garbage collection via Reference Counting
  - Swift memory model

Variables

- int
- float
- string
- bool
- List
- Map



```
// Creation
a := 10 //infer type, here int
b := a
let c: int = 10; //explicit

// Assignment
a = 20

// Lists
a := []
a += e //append e
a -= e //delete e, nur das erste
e := a[0] //access
a[0] = e // einfügen, wenn idx < a.size

// Maps
a := int -> string
a[key] = value
```

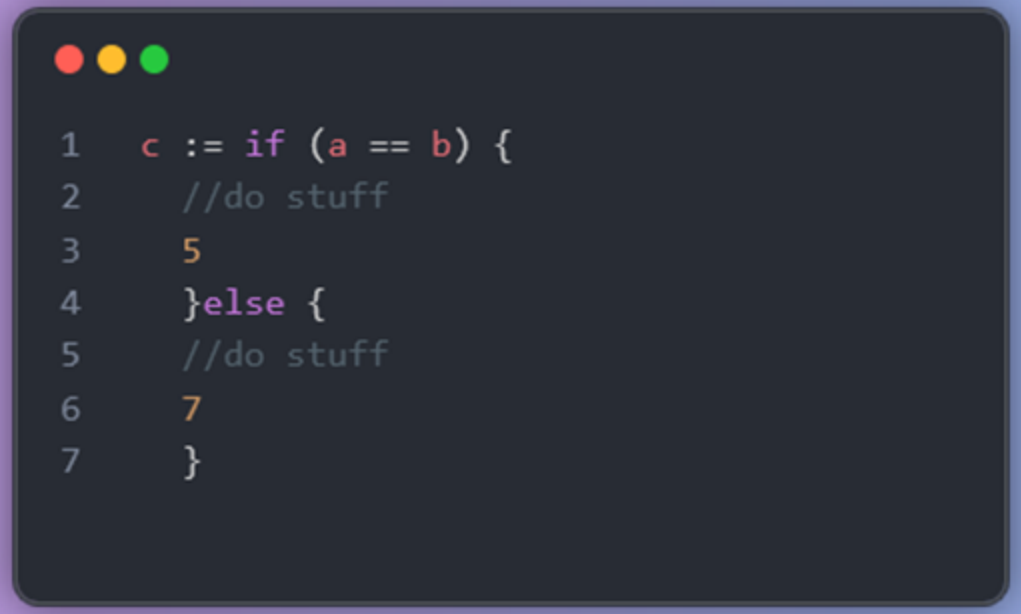
Some/None

```
let a: B? = some(10);
let b: B? = none;

let c: B = none; //error
```

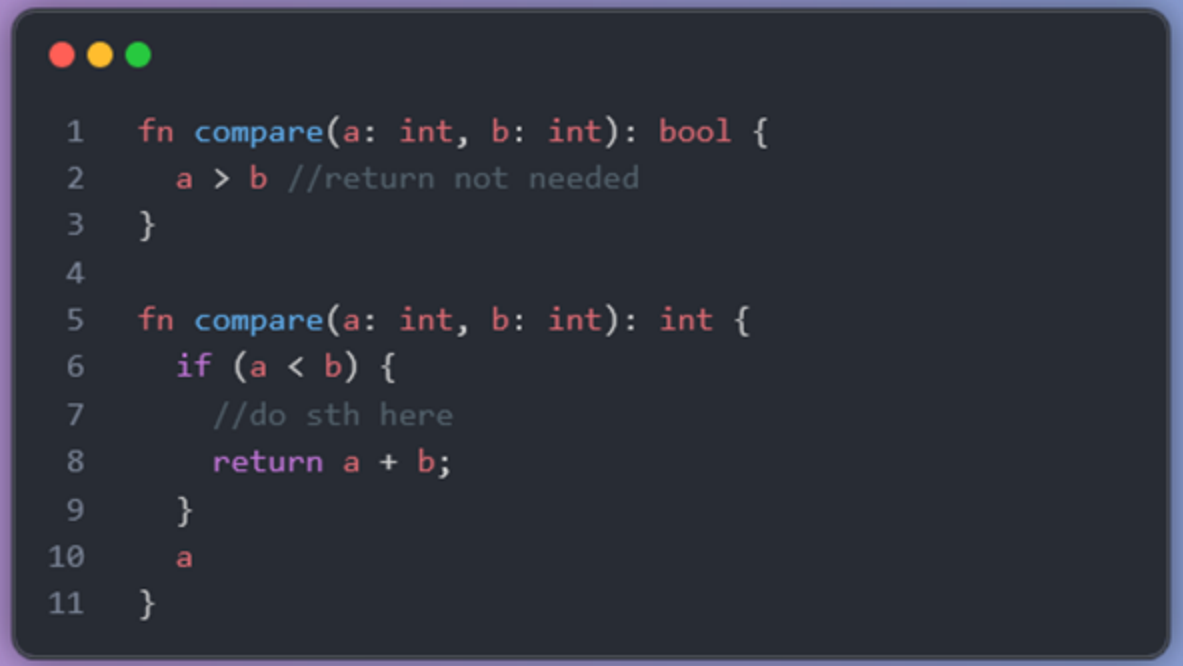
Flow Control

```
if (true){
  ...
} else if (...){
  ...
} else{
  ...
}

while (true){...}

for (i := 0; i < 10; i += 1){...}

for (a in list) {...}
```

If



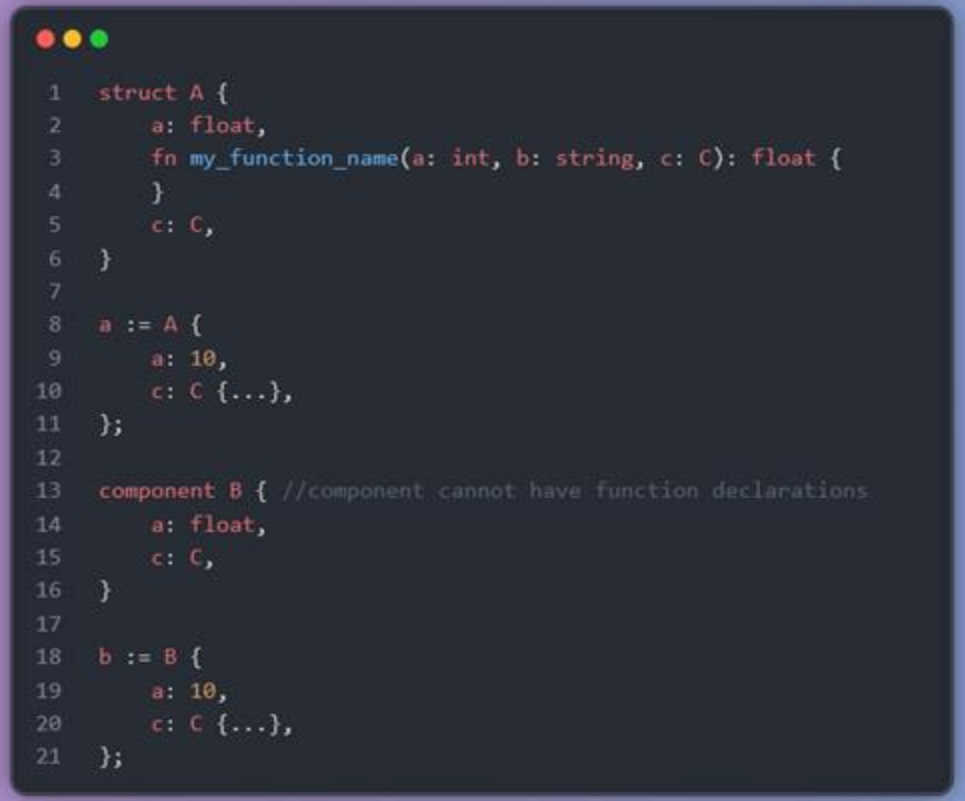
Functions



```
fn compare(a: int, b: int): bool {
  a > b //return not needed
}

fn compare(a: int, b: int): int {
  if (a < b) {
    //do sth here
    return a + b;
  }
  a
}
```

Structs and Components

- No inheritance
- Weak references



```
struct A {
    a: float,
    fn my_function_name(a: int, b: string, c: C): float {
    }
    c: C,
}

a := A {
    a: 10,
    c: C {...},
};

component B { //component cannot have function declarations
    a: float,
    c: C,
}

b := B {
    a: 10,
    c: C {...},
};
```

System

```
system s1 (a: <P1>, b: <P2>, c: <P3>)
querying
  <P1> as List with {Entity, Component1, Component2} % {!Component3}
  <P2> as List with {Entity, Component1}
                    % { #Parent: { Component2 && Component3} &&
                      Any<#Children>: {Component4} &&
                      Component6 && Component7 && !Component8 || Component9}
  <P3> as Single with {Entity, Camera} % {MainCamera} {
  [Code goes here!]
}
```

System Query

```
<P1> as List with {Entity, Component1, Component2} % {!Component3}
```

List/ Single	members of an entity	Condition

- P1 is a list where every entry consists of the entity-id, Component1 and Component2
- All entities must have Component1 and Component2
- Do not have the Component3

Register system

```
register s1;
register s2 -> s3; // s1 -> s2 -> s3
register s4 after s1; // s1 -> s4 -> s2 -> s3
register s5 before s1; // s5 -> s1 -> s4 -> s2 -> s3
unregister s2; // s5 -> s1 -> s4 -> s3


group PreUpdate {
  s1, s2, s3
}

group Update {
  s4,
  s5 -> s6,
  s6 -> s7,
  s6 -> s8,
}

register group PreUpdate -> group Update;
```

Entity

```
create entity e1
  with
    C1,
    C2,
    C3;

e1 += CameraControllerComponent {
  fov: 45.0,
  speed: 10.0,
  distanceToTarget: 10.0,
};

e1 -= CameraControllerComponent;

remove entity e1;
```

Grammar

- LR(1) – Grammar
- Lalrpop as parser generator
  - Rust
  - Supports LR(1) and LALR(1) so far

Grammar – LR(1)-Problems

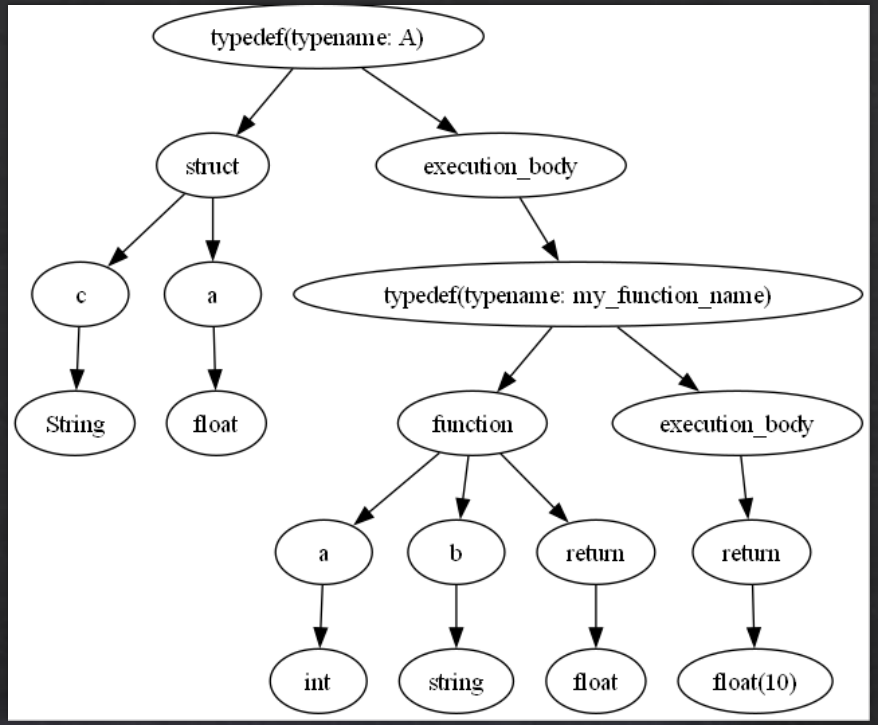
- Ambiguity when trying to support if/while/for without parenthesis
  - if a.b.c {} {} can not be parsed, due to only one look ahead
  - if (a.b.c{}){} can be parsed

  - Cannot decide whether c {} is a struct definition or the body

Current Status: MVP

AST

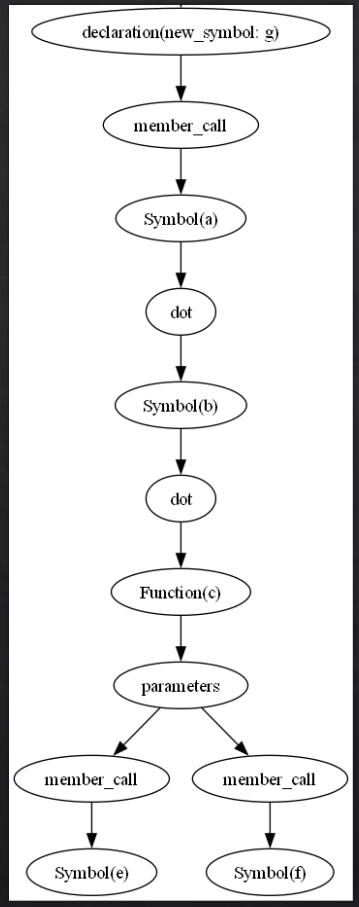
```
struct A {
    a: float,
    fn my_function_name(a: int, b: string): float {
        return 10.0;
    }
    c: String,
}
```




```
<l:@L> struct_term <name:id> l_brace <body:StructBlock> r_brace <r:@R>
  => AstNode::new(l..r, AstNodeType::TypeDef {
      typename: name, typedef: AstTypeDefinition::Struct(body.attributes), execution_body: body.functions
      })
```

Member Access

```
1   g := a.b.c(e,f);
```



Interpreter

- Naive
- Dynamically Typed

```rust
pub fn eval_node(&mut self, node: &AstNode) -> Result<IsReturn, Error> {
    let evaluated = match &node.type_of {
        // Primitives
        AstNodeType::Bool(b) => {
            IsReturn::NoReturn(InterpreterValue::new_strong(InterpreterValue::Bool(*b)))
        }
        AstNodeType::Int(i) => {
            IsReturn::NoReturn(InterpreterValue::new_strong(InterpreterValue::Int(*i)))
        }
        AstNodeType::Float(f) => {
            IsReturn::NoReturn(InterpreterValue::new_strong(InterpreterValue::Float(*f)))
        }
        AstNodeType::String(s) => IsReturn::NoReturn(InterpreterValue::new_strong(
            InterpreterValue::String(s.clone()),
        )),
        AstNodeType::List(values) => IsReturn::NoReturn(self.eval_list(values)?),
        AstNodeType::Map(values) => IsReturn::NoReturn(self.eval_map(values)?),
        AstNodeType::Symbol(s) => IsReturn::NoReturn(self.eval_symbol(s)?),
        AstNodeType::Weak(inner) => IsReturn::NoReturn(self.eval_weak(inner.as_ref())?),

        // Member call can be anything that is of the form a.b.c.d(a,b).c etc.
        // a() and a are also member calls with length 1
        AstNodeType::MemberCall { calls } => self.eval_member_call(calls)?,
        AstNodeType::ReturnStatement { return_value } => {
            IsReturn::Return(self.eval_node(return_value.as_ref())?.unwrap())
        }
        [...]
        _ => IsReturn::NoReturn(InterpreterValue::Empty),
    };

    Ok(evaluated)
}
```

Interpreter - Staging

```rust
pub enum Stages {
    Parser(Parser),
    Preprocessor(Preprocessor),
    Interpreter(Interpreter),
}

pub enum StageResult {
    PreParse(String),
    Parsing(Vec<AstNode>),
    Preprocessor(Scope, Vec<AstNode>),
    Interpretation,
}

```

Preprocessor

- No static type checking
- Simply loads all top level definitions of functions and types (structs)
- Filtering rest of AST to only contain non typedef nodes

Interpreter - Value

```rust
#[derive(Clone, Debug)]
pub enum InterpreterValue {
    Int(i64),
    Float(f64),
    String(String),
    Bool(bool),
    List(Vec<InterpreterValue>),
    Map(HashMap<InterpreterValue, InterpreterValue>),
    Struct(Symbol, HashMap<Symbol, Box<InterpreterValue>>),
    Option(Option<Box<InterpreterValue>>),
    Result(Result<Box<InterpreterValue>, Box<InterpreterValue>>),
    Function(Symbol), // Functions execution body is contained in its type definition,

    // Reference counted values (everything afaik)
    Weak(Weak<InterpreterValue>),
    Strong(Rc<InterpreterValue>),

    // ECS Intergration
    Entity(Index<Entity>),
    Component(Symbol, HashMap<Symbol, Box<InterpreterValue>>),

    // This can be any scope
    Module(Rc<RefCell<Scope>>),

    // Represents nothing, i.e. no value is returned
    Empty,
}

```

Scopes

```rust
#[derive(Debug)]
pub struct Scope {
    parent: Option<Rc<RefCell<Scope>>>,
    values: HashMap<Symbol, InterpreterValue>,
    types_for_variable: HashMap<Symbol, TypeSymbol>,
    defined_types: HashMap<Symbol, TypeSymbol>,
}

```

Problems

- Interpreter might be too slow for an ECS (Hopefully JIT might help here)
- “Communication“ between ECS and Interpreter
- Dependent on ECS implementation
- Own AST implementation
  - Some recursive node structures might need refactoring

Evaluation

- Line of code comparisons with other ECS
- Let other developers use the language and evaluate it
  - Ergonomics
  - Subjective experience
- Writing a simple game with ECSInject

What is to come

- Just In Time compilation (JIT)
  - Most systems run every frame => possible performance improvement
  - ECS integration must be implemented using an interpreter
- FFI with C libraries (one way)
- Static Type Checking
- Type Inference
- Cycle Detection
- REPL

Sources

- Lalrpop
  - https://lalrpop.github.io/lalrpop/index.html
- ECS
  - https://docs.unity3d.com/Packages/com.unity.entities@6.4/manual/concepts-intro.html
  - https://www.theknowledgeacademy.com/blog/entity-component-system/
- Bevy
  - https://bevy.org/

Recap

- Goal:
  - Statically Typed language
  - Entities, Systems and Components as first-class citizens (including queries)

- Currently:
  - Basic grammar definition and LR(1) parser
  - MVP with partially working interpreter