+ Scala = ?

Ren'Py in Scala

Paul Tan and Bing You

https://github.com/pyokagan/DokiDokiCS4215

Ren’Py

What is Ren’Py?

- A framework for writing **visual novels**.
  - Visual novels: Interactive text-based stories aided by *visuals* and *audio*.
- Implemented in Python
- DSL: **The Ren’Py Language**
  - A language tailored towards writing screenplay interspersed with game logic.

script.rpy

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

Ren’Py Script vs Stage Script

SCENE 1

The football-club locker-room. The locker-room is dark and empty. The main lights are switched on. OLD JOHN and TONY enter stage right. OLD JOHN is walking with the help of a stick.

Stage Directions

OLD JOHN New paint job is it?

T… New paint. New benches. New lockers. Even got new soap for the showers.

Character

```
label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    cene black with dissolve
    {b}Good Ending{/b}."
```

Stage Directions

Character

Say Nodes

Waits for user input before continuing with execution.

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

Image Nodes

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her ri
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

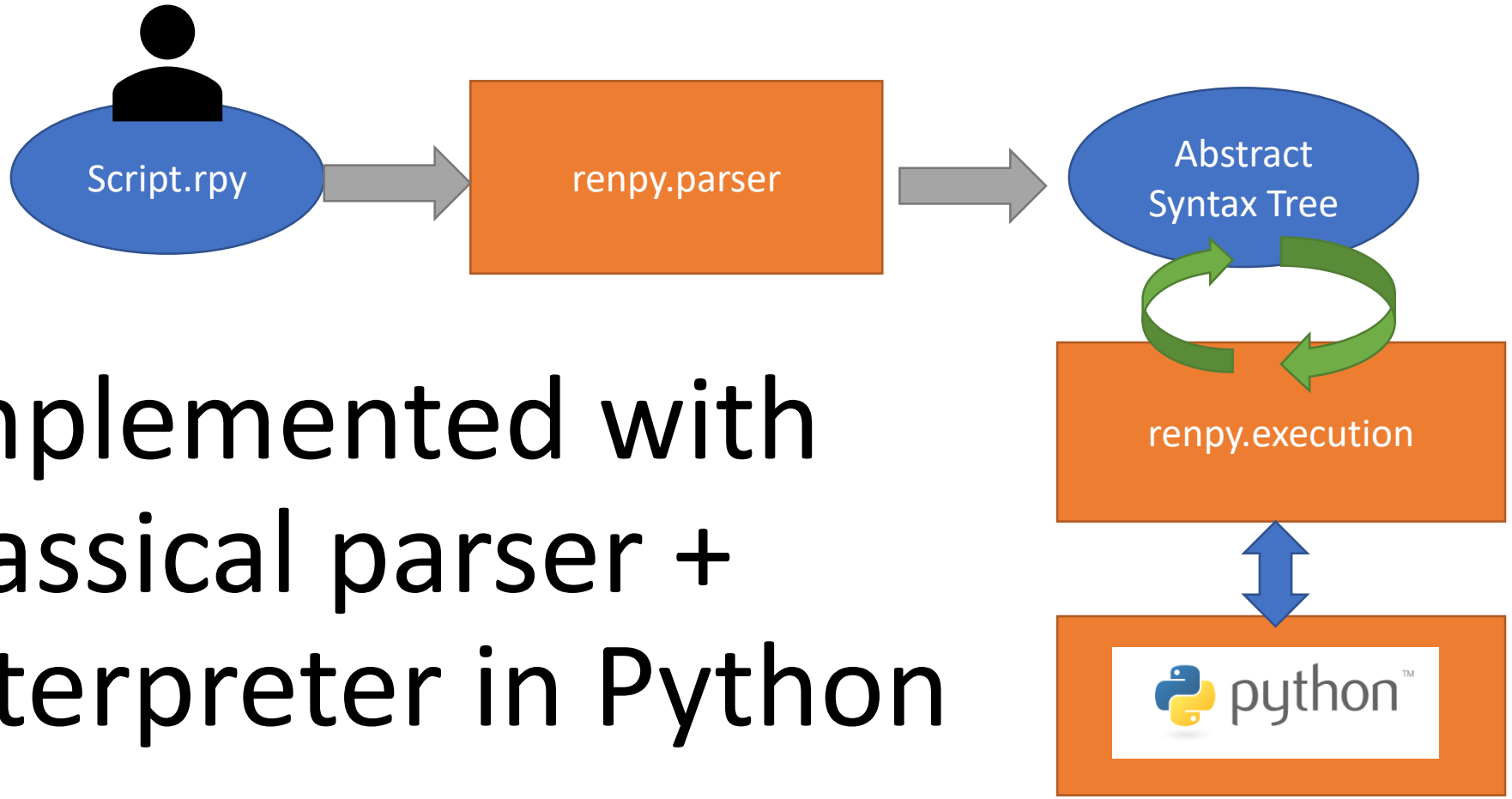
Menu Node

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her ri
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

Label Nodes

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her ri
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

Implementation



Implemented with classical parser + interpreter in Python

Now, let’s build it in Scala…

script.rpy

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

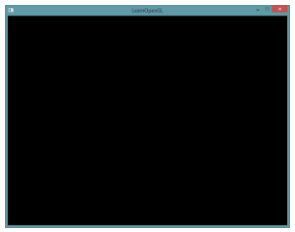
Game.scala

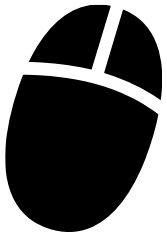

```
object Game {
  // Declare characters used by this game.
  val s = new Character("Sylvie", color = "#abcdef")
  val m = new Character("Me", color = "#c8c8ff")

  // The game starts here.
  def run(): Future[Unit] =
    scene("bg uni") |>
    "When we come out of the university, I spot her right away." |>
    show("sylvie green normal") |>
    "Sylvie's got a big heart and she's always been a good friend to me." |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))

  def rightaway(): Future[Unit] =
    show("sylvie green smile") |>
    m :< "Are you going home now? Wanna walk back with me?" |>
    s :< "Why not?" |>
    sceneBlack() |>
    "Good Ending."

  def later(): Future[Unit] =
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later." |>
    sceneBlack() |>
    "Bad Ending."
}
```

What we need…

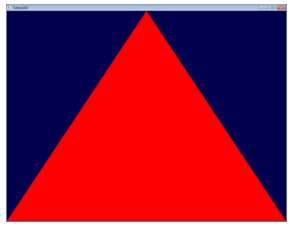
- A good implementation programming language

- Windowing / User Input



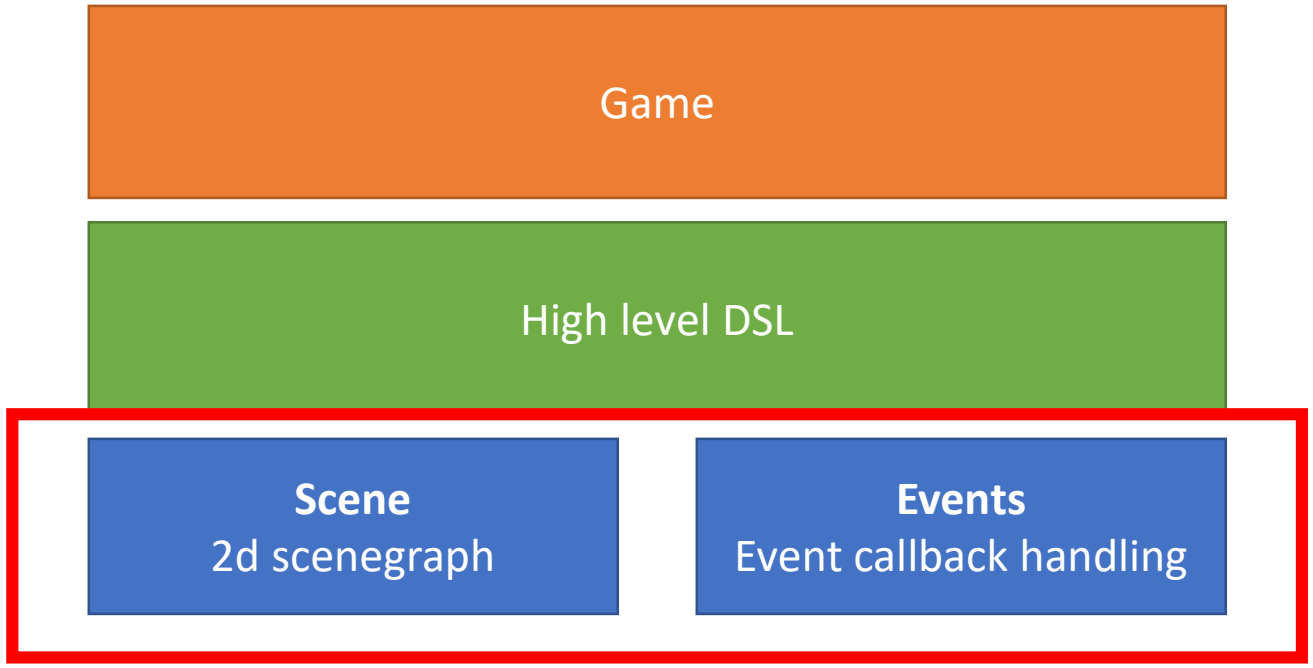


- Graphics rendering



Java bindings for GLFW, OpenGL

Application architecture

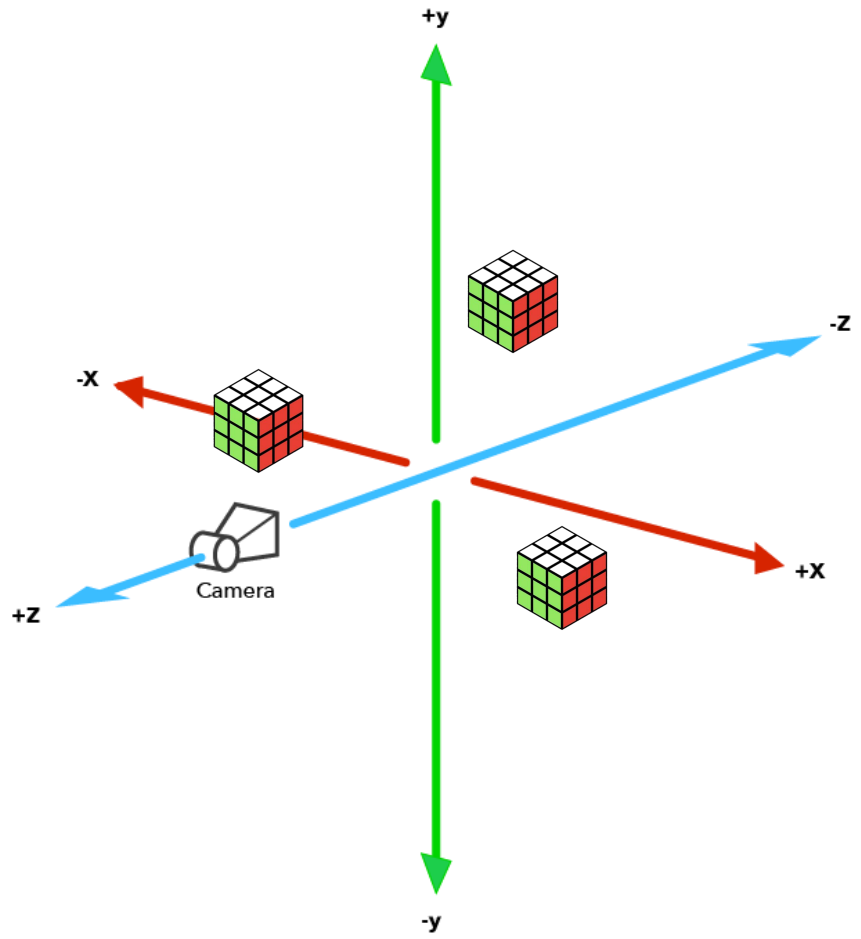


Engine subsystems

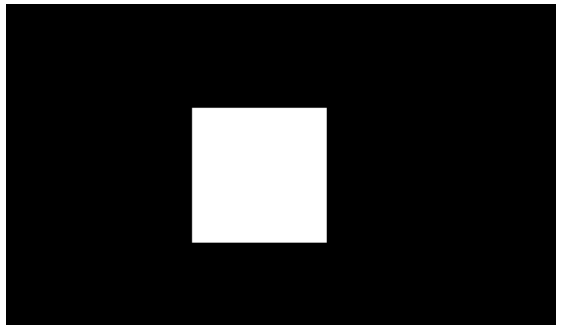
LWJGL
Lightweight Java Game Library

Scene subsystem

- Models 3d space in which *objects* (scene nodes) can be placed in a declarative manner.
- Scene library takes care of **rendering** the objects (with 2d projection) on the screen using OpenGL.
- Scene nodes have position, orientation, scale.



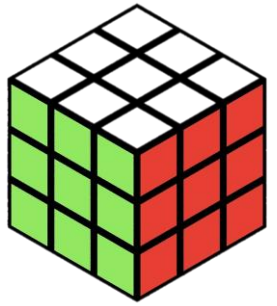
Scene nodes



```
val squareNode = new Scene.SquareNode()
squareNode.pose.scale.set(100f)
Scene += squareNode
```

```
val imageNode = new Scene.ImageNode("sylvie blue giggle")
Scene += imageNode
```

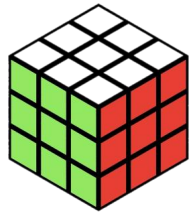

```
val textNode = new Scene.TextNode("Hello World!")
Scene += textNode
```

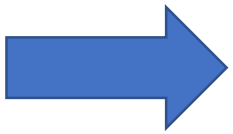


Scene Rendering

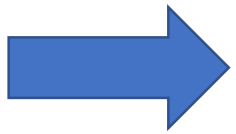
- OpenGL calls are executed based on contents of the scene graph, so your objects are rendered on screen.

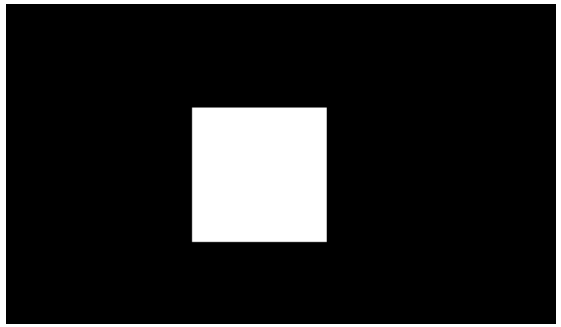
```
val squareNode = new Scene.SquareNode()
squareNode.pose.scale.set(100f)
Scene += squareNode
```





```
val vao = glGenVertexArrays()
glBindVertexArray(vao)
glBindBuffer(GL_ARRAY_BUFFER, arrayBuf)
glBufferData(GL_ARRAY_BUFFER, vertexData, GL_STATIC_DRAW)
val aPosLoc = program.getAttribLocation("aPos")
glVertexAttribPointer(aPosLoc, 2, GL_FLOAT, false, 8, 0)
glEnableVertexAttribArray(aPosLoc)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, QuadElem.glBuf)
program.use()
program.setUniform("uMVPMatrix", uMVPMatrix)
program.setUniform("uColor", uColor)
program.setUniform("uOpacity", uOpacity)
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0)
glBindVertexArray(0)
```





Events

- The Events subsystem maintains a set of callbacks for each event type

```
object Events {
  val onKeyPress = scala.collection.mutable.HashSet.empty[Key => Unit]
  val onMouseButtonPress = scala.collection.mutable.HashSet.empty[MouseButton => Unit]
  val onTick = scala.collection.mutable.HashSet.empty[() => Unit]
  ...
```

- E.g. to register a event handler:

```
Events.onKeyPress += (key => println("Key was pressed: " + key))
```

Events – Game loop

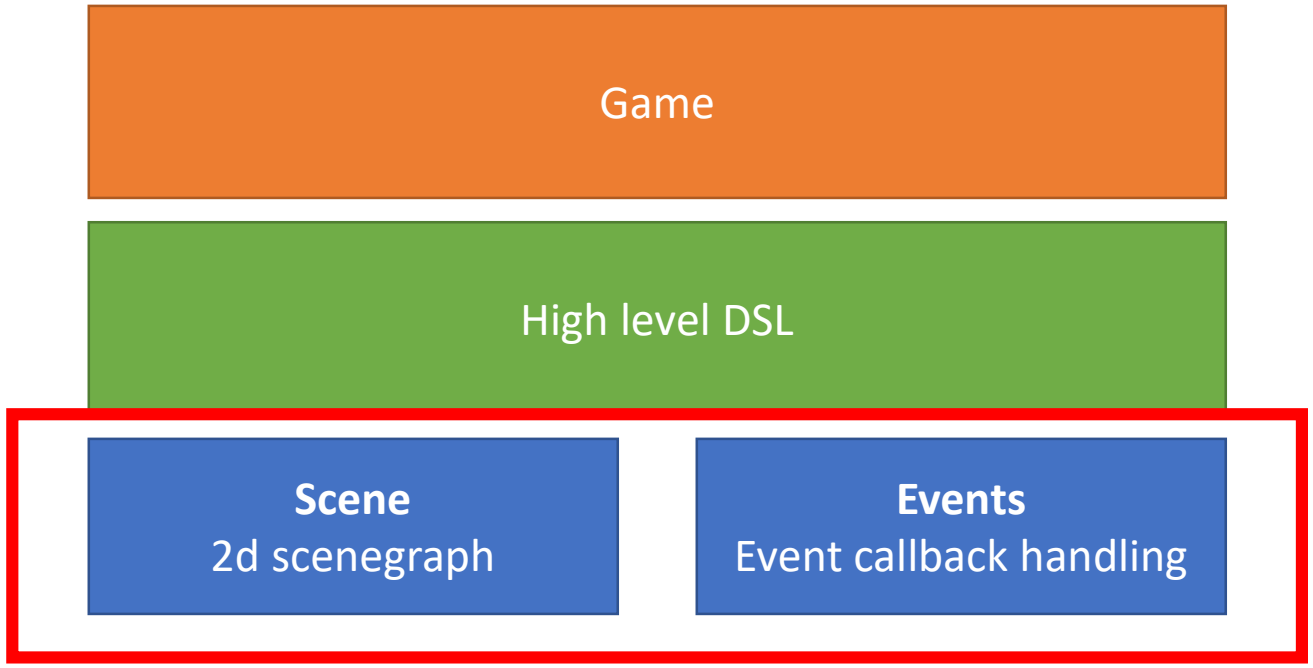
- The Events subsystem also manages the **event loop**, which is the central control flow construct of the program

```
while (!glfwWindowShouldClose(window)) {
    Events.runCallbacks()

    // Logic ticks
    val currentTime = glfwGetTime()
    while (lastTick + TickPeriod < currentTime) {
      Events.tick()
      lastTick += TickPeriod
    }

    // Render
    Scene.render()
    glfwSwapBuffers(window)
    glfwPollEvents()
}
```

Virtually an infinite loop – runs for the entire lifetime of the program (until the user closes the window)

Calls onKeyPress callbacks on key press, onMouseButtonPress callbacks when mouse button is pressed, etc.
Must be called periodically to process user input in a timely manner.

Application architecture



Engine subsystems

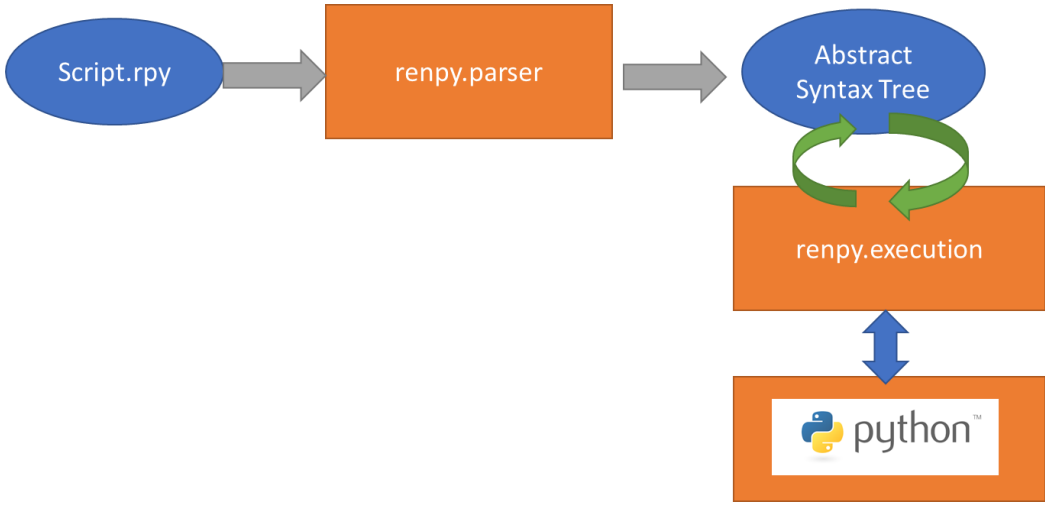
LWJGL
Lightweight Java Game Library

Type of DSL Implementations

- Standalone DSLs
- Deeply-embedded DSLs
- Shallowly-embedded DSLs

Standalone DSL

- What RenPy did: separate parser and interpreter
- Lots of work to implement
  - IDE, tooling support, syntax highlighting, debuggers…



Deep embeddings of DSLs

- Construct an AST.
- AST is then *traversed* for evaluation.

Benefits:

- Can arbitrarily transform the AST for e.g. optimization, implementing complex non-compositional semantics

Cons:

- More implementation work needed

$$\begin{aligned}
&\textbf{data } Expr :: * \textbf{ where} \\
&\quad Val \;\; :: Integer \to Expr \\
&\quad Add :: Expr \to Expr \to Expr
\end{aligned}$$

$$\begin{aligned}
&eval :: Expr \to Integer \\
&eval\ (Val\ n) \quad\;\; = n \\
&eval\ (Add\ x\ y) = eval\ x + eval\ y
\end{aligned}$$

$$3 + 4 \longrightarrow Add\ (Val\ 3)\ (Val\ 4)$$

Shallow embeddings of DSLs

- No AST constructed – terms are immediately evaluated

Benefits:

- **Implementation is trivial and compact**

Cons:

- Not as flexible – semantics must be compositional (depends only on semantics of components)

$$\textbf{type } Expr = Integer$$
$$val :: Integer \rightarrow Expr$$
$$val\ n \quad = n$$
$$add :: Expr \rightarrow Expr \rightarrow Expr$$
$$add\ x\ y = x + y$$

We aren't trying to do anything fancy, so let's try implementing it using shallow embedding.

Overall Approach

- Use **shallow embedding** for implementation.
  - Implement DSL syntax directly using evaluation functions
- Desired semantics: *Asynchronous Actions*
  - We want to have different actions occurring concurrently (e.g. animation)
  - At the same time, some actions need to wait on each other (e.g. wait for user input)
  - Use existing Scala infrastructure: **Futures and Promises**

DSL API Design:

```
object Dsl {
  case class Character(name: String, color: String)

  // Say a message and wait for user input
  def say(char: Character, msg: String): Future[Unit]
  def say(msg: String): Future[Unit]

  // Show a background image
  def show(image: Texture): Future[Unit]

  // Ask the user to make a choice. Execute the associated function and wait on its
completion.
  def menu(msg: String, choices: (String, () => Future[Unit])): Future[Unit]
}
```

How to implement say(), show(), menu()?

Futures and Promises

- Future: Read-only placeholder containing a value that may not yet exist.
- Promise: Writable container that completes a Future

Futures and Promises

```
val p = Promise[T]()
val f = p.future

val producer = Future {
 val r = createResource()
  p success r
  continueProducerExecution()
}

val consumer = Future {
  startConsumerExecution()
  f foreach { r =>
    consumeResource()
  }
}
```

(1) Create Promise containing Future

(2) Run *producer* and *consumer* asynchronously

(3) *producer* creates resource and continues execution

(4) *consumer* use up resource and continues execution

producer execution is not blocked by consumer

consumer can receive resource before *producer* finishes execution

Implementation building blocks

- Create fundamental asynchronous execution functions based on futures and promises
  - waitForMouseButtonPress(), nextEvent()
- Build DSL functions based on said functions
  - say(), show(), scene(), menu()

waitForMouseButton()

```
def waitForMouseButtonPress()(implicit ec: Ex
  val promise = Promise[Unit]()
  val cb = new Function1[MouseButton, Unit] {
    def apply(mouseButton: MouseButton): Unit = {
      if (mouseButton == MouseButtonLeft) {
        Events.onMouseButtonPress -= this
        promise.success(mouseButton)
      }
    }
  }
  Events.onMouseButtonPress += cb
  promise.future
}
```

(1) Create promise containing Future that will be completed later.

(4) Remove mouse button press callback from our events system

(5) Complete the Future

(2) Register mouse button press callback with our events system

(3) Return the Future

```
val charName = new Scene.TextNode {
  maxWidth = 700f / 0.28f
  pose.position.set(-380f, -205f, 20f)
  pose.scale.set(0.28f, 0.28f, 1f)
}
```

```
val msgNode = new Scene.TextNode {
  maxWidth = 700.0f / 0.25f
  pose.position.set(-360f, -250f, 20f)
  pose.scale.set(0.25f, 0.25f, 1f)
}
```

say()

```
val textbox = new Scene.ImageNode("textbox") {
  pose.position.set(0f, -268f, 10f)
}
```

say()

```
def say(char: Character, msg: String)(implicit e                    = {
  Scene ++= Seq(textbox, charName, msgNode)
  msgNode.text = msg
  charName.text = char.name
  charName.color = parseHexColour(char.colo
  Events.waitForMouseButtonPress().flatMap(_ => {
    Scene --= Seq(textbox, charName, msgNode)
    Events.nextEvent()
  })
}
```

(1) Add textbox, charName, msgNode to Scene graph so that they will be displayed, and set the msgNode text, charName text and color

(2) Wait for left mouse button press

(3) Cleanup: Remove textbox, charName, msgNode from scene graph so they are not displayed any more.

Sequencing Futures with flatMap

- Future provides a `flatMap()` method allowing us to sequence asynchronous operations.

```
def flatMap[S](f: (T) ⇒ Future[S]): Future[S]
```

Creates a new future by applying a function to the successful result of this future, and returns the result of the function as the new future.

```
say("a").flatMap(() => say("b"))
```

Say “a”, wait for user input, and then say “b”.

Problem: Lots of flatMaps…

```
def run(): Future[Unit] =
   scene("bg uni") flatMap
   ( _ => say("When we come out of the university, I spot her right away.")) flatMap
   ( _ => show("sylvie green normal")) flatMap
   ( _ => say("Sylvie's got a big heart and she's always been a good friend to me.")) flatMap
   ( _ => menu("As soon as she catches my eye, I decide...",
     ("To ask her right away.", rightaway),
     ("To ask her later.", later)))
```

```
# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later
```

Problem: Lots of flatMaps…

```
def run(): Future[Unit] =
    scene("bg uni") flatMap
    ( _ => say("When we come out of the university, I spot her right away.")) flatMap
    ( _ => show("sylvie green normal")) flatMap
    ( _ => say("Sylvie's got a big heart and she's always been a good friend to me.")) flatMap
    ( _ => menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later)))
```

```
# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later
```

Solution: Use Scala implicit classes to implement a |> infix operator

```
object Dsl {
  ...
  implicit class FutureWith[T](a: Future[T]) {
    def |>[S](b: => Future[S]): Future[S] =
      a.flatMap(_ => b)
  }
  ...
}
```

Use By-Name parameter -- Only evaluate RHS once the LHS has completed.

|> starts with | which has the lowest precedence (for symbol infix operators) in Scala

```
def run(): Future[Unit] =
    scene("bg uni") |>
    say("When we come out of the university, I spot her right away.") |>
    show("sylvie green normal") |>
    say("Sylvie's got a big heart and she's always been a good friend to me.") |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))
```

Problem: say(), say(), say(), say()….


```
def run(): Future[Unit] =
    scene("bg uni") |>
    say("When we come out of the university, I spot her right away.") |>
    show("sylvie green normal") |>
    say("Sylvie's got a big heart and she's always been a good friend to me.") |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))
```

```
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later
```

Problem: say(), say(), say(), say()….

```
def run(): Future[Unit] =
    scene("bg uni") |>
    say("When we come out of the university, I spot her right away.") |>
    show("sylvie green normal") |>
    say("Sylvie's got a big heart and she's always been a good friend to me.") |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))
```

```
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later
```

Must use say() in Scala but just plain strings will do in Ren’Py – Scala version has more syntactic overhead

Solution: Use Scala implicit classes (again) to implement |> infix operator for Strings

```
object Dsl {
  ...
  implicit class FutureWithDsl[T](a: Future[T]) {
    def |>[S](b: => Future[S]): Future[S] =
      a.flatMap(_ => b)
    def |>(b: String): Future[Unit] =
      a.flatMap(_ => say(b))
  }

  implicit class StringWithDsl[T](a: String) {
    def |>[S](b: => Future[S]): Future[S] =
      say(a).flatMap(_ => b)
    def |>(b: String): Future[Unit] =
      say(a).flatMap(_ => say(b))
  }
  ...
}
```

Solution: Use Scala implicit classes (again) to implement |> infix operator for Strings

```
 def run(): Future[Unit] =
    scene("bg uni") |>
    "When we come out of the university, I spot her right away." |>
    show("sylvie green normal") |>
    "Sylvie's got a big heart and she's always been a good friend to me." |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))
```

```
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later
```

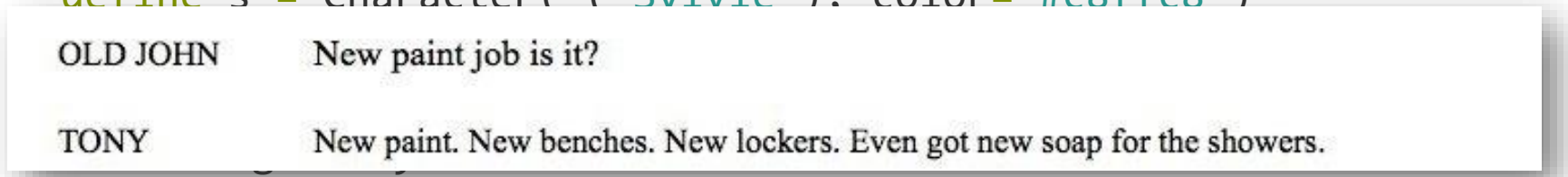
Problem: say() with characters

```
val s = new Character("Sylvie", color = "#abcdef")
val m = new Character("Me", color = "#c8c8ff")

def rightaway(): Future[Unit] =
  show("sylvie green smile") |>
  say(m, "Are you going home now? Wanna walk back with me?") |>
  say(s, "Why not?") |>
  sceneBlack() |>
  "Good Ending."
```


```
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return
```

Problem: say() with characters

```
val s = new Character("Sylvie", color = "#abcdef")
val m = new Character("Me", color = "#c8c8ff")

def rightaway(): Future[Unit] =
  show("sylvie green smile") |>
  say(m, "Are you going home now? Wanna walk back with me?") |>
  say(s, "Why not?") |>
  sceneBlack() |>
  "Good Ending."
```




```
define s = Character( ("Sylvie"), color="#c8ffc8")
show sylvie green smile
m "Are you going home now? Wanna walk back with me?"
s "Why not?"
scene black with dissolve
"{b}Good Ending{/b}."
return
```

Solution: Implement :< infix operator for characters

- Scala's syntax doesn't allow us to implement Ren'Py's exact syntax, but we can get close by defining a lightweight infix operator :<

```
case class Character(name: String, color: String) {
    def :<(m: String): Future[Unit] =
      say(this, m)
  }
```

```
def rightaway(): Future[Unit] =
    show("sylvie green smile") |>
    m :< "Are you going home now? Wanna walk back with me?" |>
    s :< "Why not?" |>
    sceneBlack() |>
    "Good Ending."
```

Wrapping up

- Scala doesn't allow us to implement Ren'Py's syntax directly, however with generous use of infix operators we can get pretty close to the spirit of a stage play script.

SCENE 1

The football-club locker-room. The locker-room is dark and empty. The main lights are switched on. OLD JOHN and TONY enter stage right. OLD JOHN is walking with the help of a stick.

OLD JOHN New paint job is it?

TONY New paint. New benches. New lockers. Even got new soap for the showers.

```
def rightaway(): Future[Unit] =
  show("sylvie green smile") |>
  m :< "Are you going home now? Wanna walk back with me?" |>
  s :< "Why not?" |>
  sceneBlack() |>
  "Good Ending."
```

Stage directions use standard function call syntax so that they stand out.

Characters lined up on the left

Light-weight syntax for text (great for text-heavy novels).

Conclusion

- We set out on a grand adventure to re-implement the RenPy DSL in the best programming language in the world (Scala)
- To do that, we first implemented a simple **2d rendering and event engine** using LWJGL.
- Then, we used the **shallow embedding** implementation approach to implement our *high level DSL.*
- Used Scala's **Futures and Promises** to implement our DSL's *async semantics.*
- Gratuitously used Scala's **implicit classes** to define our own **infix operators** that can be used with existing Scala types.

script.rpy

```
# Declare characters used by this game.
define s = Character(_("Sylvie"), color="#c8ffc8")
define m = Character(_("Me"), color="#c8c8ff")

# The game starts here.
label start:
    scene bg uni with fade
    "When we come out of the university, I spot her right away."
    show sylvie green normal with dissolve
    “Sylvie’s got a big heart and she's always been a good friend to me."
    menu:
        "As soon as she catches my eye, I decide..."
        "To ask her right away.":
            jump rightaway
        "To ask her later.":
            jump later

label rightaway:
    show sylvie green smile
    m "Are you going home now? Wanna walk back with me?"
    s "Why not?"
    scene black with dissolve
    "{b}Good Ending{/b}."
    return

label later:
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later."
    scene black with dissolve
    "{b}Bad Ending{/b}."
    return
```

Game.scala


```
object Game {
  // Declare characters used by this game.
  val s = new Character("Sylvie", color = "#abcdef")
  val m = new Character("Me", color = "#c8c8ff")

  // The game starts here.
  def run(): Future[Unit] =
    scene("bg uni") |>
    "When we come out of the university, I spot her right away." |>
    show("sylvie green normal") |>
    "Sylvie's got a big heart and she's always been a good friend to me." |>
    menu("As soon as she catches my eye, I decide...",
      ("To ask her right away.", rightaway),
      ("To ask her later.", later))

  def rightaway(): Future[Unit] =
    show("sylvie green smile") |>
    m :< "Are you going home now? Wanna walk back with me?" |>
    s :< "Why not?" |>
    sceneBlack() |>
    "Good Ending."

  def later(): Future[Unit] =
    "I can't get up the nerve to ask right now. With a gulp, I decide to ask her later." |>
    sceneBlack() |>
    "Bad Ending."
}
```