Hello Godot

Nodes – building blocks

Scenes – Objects

Scripts need nodes

# comments

Extends = class

Func \_ready():

Pass

When brought into scene, do nothing

Ctrl + shift + F11 = expand view

Dynamic language like Python

Use tabs for indicating code blocks

Print() = console log

Ctrl + click to get info

Engine figures out data type

Var person = “you”

Editor/Editor Settings/Themes

Editor/Editor Settings/Shortcuts

Section 2: Loony Lips

Planning Loony Lips

Focus on core competency

Loony Lips like Mad Libs

Give prompts, fill out prompts, read story with the prompts inputted

Chalkboard background

Introducing Arrays

Use variables to make game scalable

Place input of words into array

Node names = PascalCase

User Interface anchors, margins adapt pos and size relative to parent

Vars may live outside of functions

Array = [bool, str, num]

Arrays are scalable

Index 0

%s = insert string

Print(story % prompts)

% used to insert next item in array (modulus)

All arguments must be used, params must have same amount of inputs

Label – Showing Text to the Player

Import resources

Import tab next to scene

TextureRect Node

Can define properties of Nodes in Inspector

Can click and drag resources into Inspector

Expand = on – texture scales to fit its bounding rectangle

Godot doesn’t refresh UI right away – go to new scene and come back

Use View/Show Helpers to see corners

Layout/Full Rect

Set default window – Project/Settings/General/Display/Window

Can set to resizable

Label Node

Can set Custom Fonts

New Dynamic Font

Click on Dynamic Font to open settings

For margin: -num for left or up

Align and Valign

Word wrapping

get\_node(“NodeName”):

Can pass in any node

Can click and drag node into params

Can alias node out

$Label

Can target properties of nodes

$Label.text

RichTextLabels can have Bb Code – used to set specific words/letters with different attributes

Ctrl + K = comment out

LineEdit – Getting Text from the Player

LineEdit vs TextEdit

LineEdit = single line of text

TextEdit = fill out multiple lines

HBox/VBoxContainer

Rows, columns

Containers – child organized by how they appear in scene tree

Can set alignment

Measurement always in pixels

$VBoxContainer/DisplayText

Signals under Node

Similar to sockets – emit signal, receiver looking for that specific signal

Can look up documentation by right-clicking node/Open Documentation

Can view all methods, signals

Must connect to script

Double-click/connect to script

PlayerText.clear()

Order of arguments matters

Buttons!

Button vs TextureButton

TextureButton can have different textures depending on buttonState

Size Flags for labels – determine scaling

To fill button – define min size, TextureButton Expand = On

For BoxContainer – Custom Constants – sets spacing between nodes

Appending Arrays and If Conditions

Scoped variables

var\_name = []

func function\_name():

Godot runs script from top down

onready var PlayerText = $VBoxContainer/HBoxContainer/PlayerText

onready waits until element is loaded

array.append: = .push()

array.size() = size of array

return

if condition\_is\_met:

elif: = else if

else:

queue\_free() and reload\_current\_scene()

Once something doesn’t have to be in the game, remove it from the game

.queue\_free() = removes from memory

If not = !=

Get\_tree() - grabs entire scene tree

Get\_tree(),reload\_current\_scene() – reloads scene

.grab\_focus() – sets control for input

Dictionaries – Adding a Story Template

Var My\_dictionary = { “key” : value,

“key”: value

}

Dictionaries are unsorted

Can have array of dictionaries

randi() % num – random int

Array.size() = array.length

randomize()

StoryBook Option 1: Story Objects

Separate data from script

Make a node, contains children for data

Ctrl + D – duplicate

Export var allows var to be used by Inspector

Can write typed script

Var prompts : PoolStringArray = []

PoolStringArray array of strings only

Var story : String

Node.get\_child\_count() gets number of children

Node.get\_child(num)

May need to define var type depending on what kind of data you’re working with

e.g. var story = {}

Player can’t add stories with this option

Hoppy Days

Planning Hoppy Days

No player attacks

Limited lives

Fast movement, high jumps

Get more likes w/ enough coins

PhysicsBody2D – Making a Character

PhysicsBody2D – any object that interacts w/ physics engine

3 choices

StaticBody2D

not designed to move

Good for walls, floors, platforms, etc.

Can have simple velocity applied to it (like conveyor belt)

RigidBody2D

Controlled by 2D physics engine

Built in behaviors for things like gravity, friction

Not controlled directly

KinematicBody2D

Meant to be player controlled

Not affected by 2D physics engine

Must have collisions

Sprites have texture

CollisionShape2D

Choose a shape

Can lock children to parents

Delta

Time in seconds between frames

Ignores jumps in framerate changes

Input map

Project/project settings

Input – built in command

If Input.is\_action\_pressed(“action”):

Pass

move\_and\_collide()

When you hit something, stop

Can get collision info on whatever it hits

Does automatically use delta

Move\_and\_slide()

When you hit something, try and move along it

Can detect floors, walls, ceilings

Automatically use delta when moving

Vector2(x,y)

Represents position in 2D space

And, or, not

Not = !=

And not

Or not

Make Bunny Jump

Have a scene to jump in

Have something to jump on

Define gravity

Define jump speed

Scenes

Any self-contained collection of assets brought together

Bring in player into level scene

Apply gravity to player for each frame

Engine calculates from top left

Apply gravity if not on the ground

Move\_and\_slide

Is\_on\_floor()

Part of move\_and\_slide()