Introduction to Godot and the GDScript programming language

CS-330: Introduction to Game Programming

What is Godot?

- Godot is an open-source game engine which supports 2D and 3D game development on a numerous computing platforms and game consoles.
- Godot's features include:
  - ✓ 2D and 3D physics engines,
  - ✓ A scripting workspace,
  - ✓ Animation editor,
  - ✓ Tilemap editor,
  - ✓ Shader editor,
  - ✓ Debugger,
  - ✓ Profiler.
- In essence, Godot is an Integrated Development Environment (IDE) for game development.
- Godot also features extensive online documentation and beginner tutorials.

What is Godot?

- Godot game engine can be downloaded from https://godotengine.org/
- Godot game engine documentation can is located at https://docs.godotengine.org/en/stable/

Why do I like Godot?

- Godot is cross-platform compatible. The Godot game engine runs on Windows, Apple (OSX), and GNU/Linux operating systems.
- Godot uses open standards for graphics: Vulkan and OpenGL.
- Godot is distributed as open-source under the MIT License model.
- Game developers may release and profit from their own original games using the Godot game engine at no cost.
- If you plan to release a game under Unity (a competitor to Godot), consider the pricing model of Unity:

https://www.polygon.com/23885373/unity-technologies-install-fee-pricing-change

Why do I like Godot (internal math stuff)

- Godot uses quaternions for 3D transformation matrices.
- This is a very good idea since Euler (pronounced “oiler”) angles are prone to gimbal lock when computing rotation of a rigid body in three dimensions (i.e. yaw, pitch, roll).
- Quaternions are impervious to gimble lock since quaternions effectively utilize a 4-tuple (i.e. W, X, Y, Z) to represent object transformations in three-dimensional space (i.e. X, Y, Z).
- To learn more about gimbal lock:

https://www.youtube.com/watch?v=z3dDsz4f20A

- Gimbal Lock and NASA’s Apollo 13 (gimbal lock is a real problem):

https://www.youtube.com/watch?v=OmCzZ-D8Wdk

Programming in Godot

- Godot natively supports a Python-like scripting language called GDScript.
- Godot also supports C# but you will need to install extra packages such as the Microsoft .NET System Development Kit (SDK) to gain this functionality.
- Godot also supports GDExtension which enables one to write add-on modules for your game in C/C++. This could be useful if you wish to add features not present in Godot.

Programming in Godot: My recommendations

- Since this this an introductory class, I *strongly* recommend programming in GDScript.
- GDScript is easy to learn and adequate for the types of 2D games I anticipate students will build in this class.
- A GDScript interpreter is built-in to Godot.

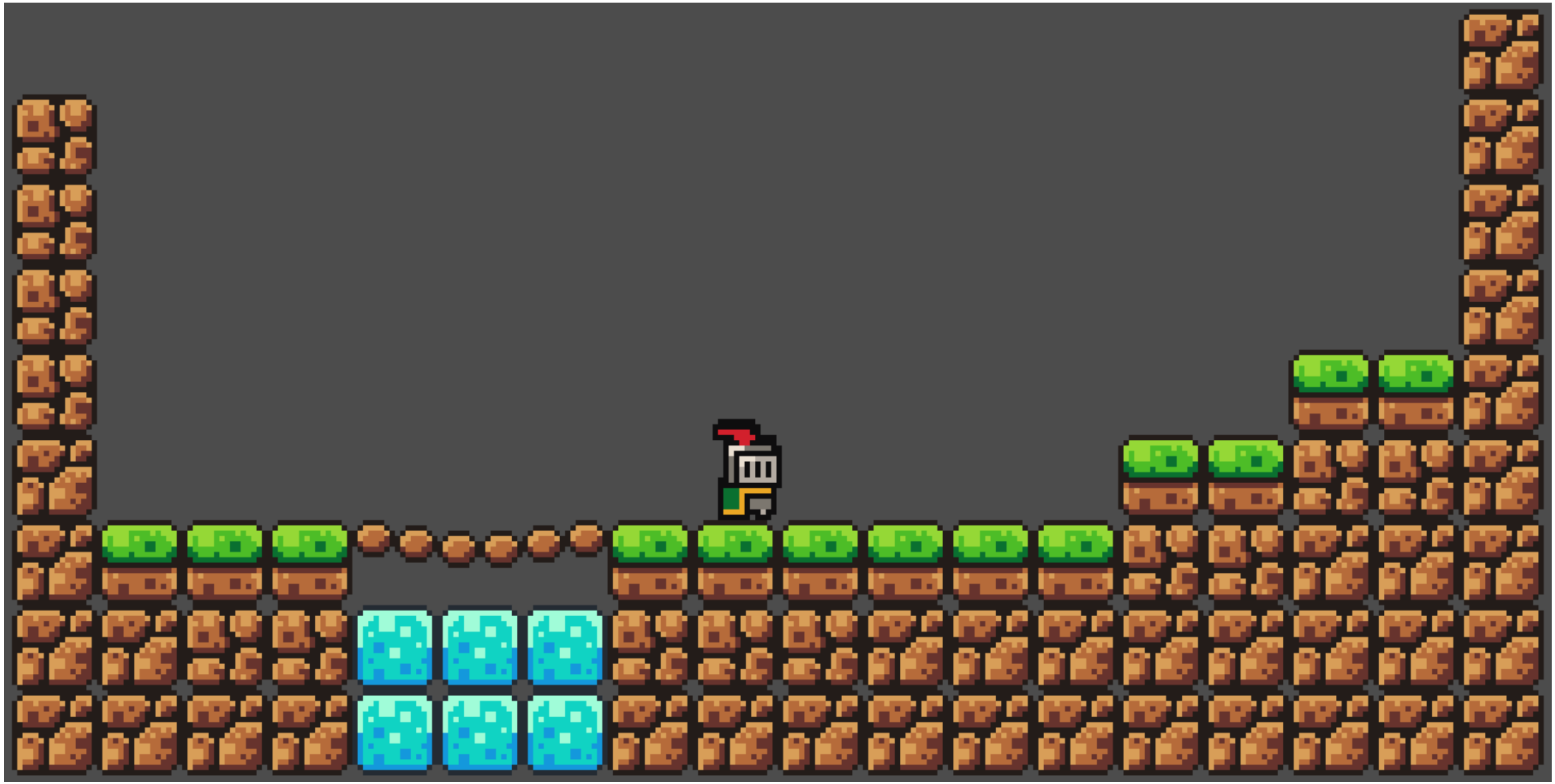
Programming in Godot

- Event-driven programming is a programming model in which code is executed based on external events.
- When you write your game logic in Godot, you will be writing programming modules to enable functionality for each feature in your game. For example, when the player hits the spacebar in a your Godot game, this event may execute a program module which makes the player’s character jump.
- Programming in Godot is very much event-driven.

What is the effect of event-driven programming on your game?

- Game programming tends to be modular.
- Each module represents some dedicated functionality specific to the game.

Work-in-progress 2D game in Godot



GDScript for a work-in-progress 2D game

```
extends CharacterBody2D
const SPEED = 130.0
const JUMP_VELOCITY = -300.0

# Get the gravity from the project settings to be synced with RigidBody nodes.
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
	# Add the gravity.
	if not is_on_floor():
		velocity.y += gravity * delta

	# Handle jump.
	if Input.is_action_just_pressed("ui_accept") and is_on_floor():
		velocity.y = JUMP_VELOCITY

	# Get the input direction and handle the movement/deceleration.
	# As good practice, you should replace UI actions with custom gameplay actions.
	var direction = Input.get_axis("ui_left", "ui_right")
	if direction:
		velocity.x = direction * SPEED
	else:
		velocity.x = move_toward(velocity.x, 0, SPEED)

	move_and_slide()
```

How do I know what key is mapped to the event “ui_accept”?

```
extends CharacterBody2D
const SPEED = 130.0
const JUMP_VELOCITY = -300.0

# Get the gravity from the project settings to be synced with RigidBody nodes.
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
	# Add the gravity.
	if not is_on_floor():
		velocity.y += gravity * delta

	# Handle jump.
	if Input.is_action_just_pressed("ui_accept") and is_on_floor():
		velocity.y = JUMP_VELOCITY

	# Get the input direction and handle the movement/deceleration.
	# As good practice, you should replace UI actions with custom gameplay actions.
	var direction = Input.get_axis("ui_left", "ui_right")
	if direction:
		velocity.x = direction * SPEED
	else:
		velocity.x = move_toward(velocity.x, 0, SPEED)

	move_and_slide()
```

How do I know what key is mapped to the event “ui_accept”?

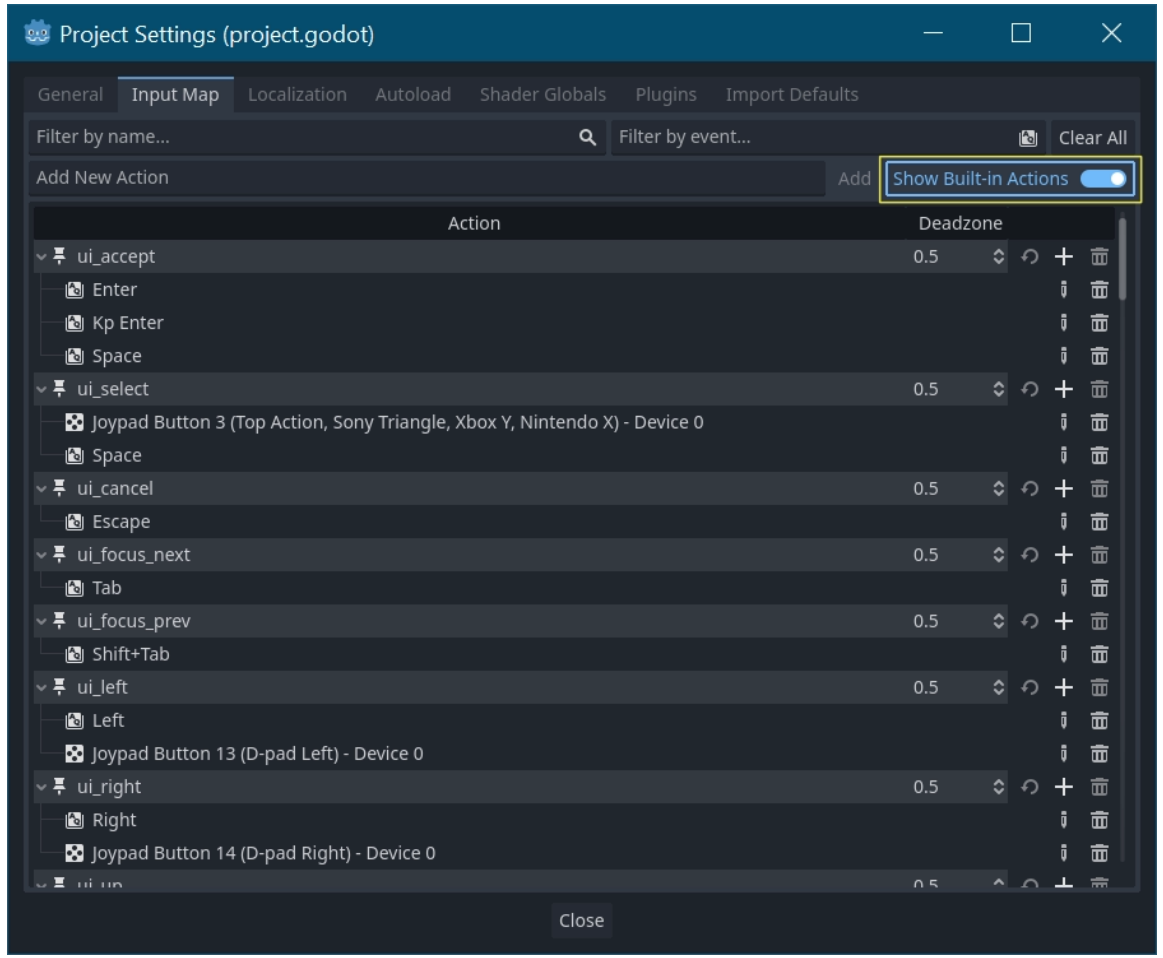
- How do I know what the key-mapping is when I see the following command (highlighted in yellow)?
  `if Input.is_action_just_pressed("ui_accept") and is_on_floor():`

- The following Godot documentation page provides some insight at:
  https://docs.godotengine.org/en/stable/tutorials/inputs/input_examples.html

- Following the link above the Godot documentation mentions input maps where keyboard, mouse, or other sorts of player inputs are mapped to Godot. To see the keyboard map:
  1. Open your Project
  2. Select Project Settings
  3. Select the Input Map tab

How do I know what key is mapped to the event “ui_accept”?

This screenshot shows that the "space" key is mapped to "ui_accept“. The enter key is also mapped to “ui_accept”. Therefore the player can press either of these keys to achieve the same functionality.

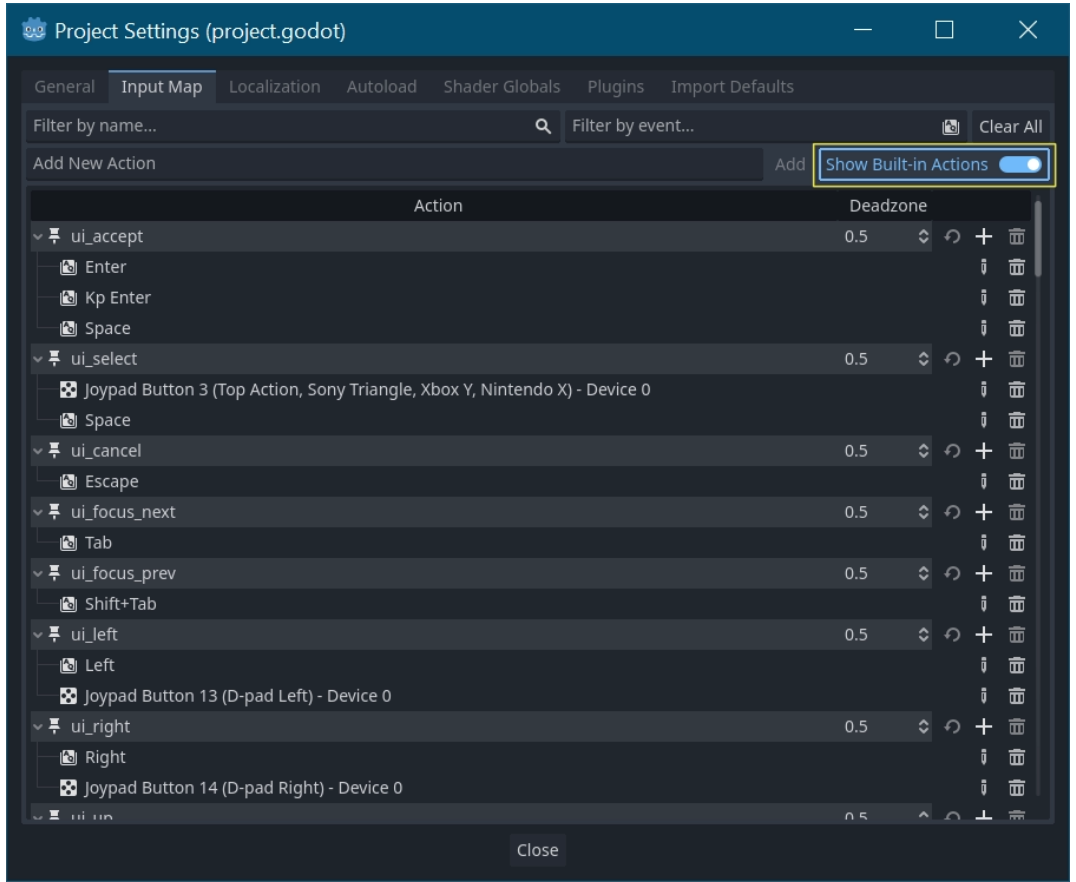


How do I know what keys “ui_left” and “ui_right” are mapped to?

```
extends CharacterBody2D
const SPEED = 130.0
const JUMP_VELOCITY = -300.0

# Get the gravity from the project settings to be synced with RigidBody nodes.
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
	# Add the gravity.
	if not is_on_floor():
		velocity.y += gravity * delta

	# Handle jump.
	if Input.is_action_just_pressed("ui_accept") and is_on_floor():
		velocity.y = JUMP_VELOCITY

	# Get the input direction and handle the movement/deceleration.
	# As good practice, you should replace UI actions with custom gameplay actions.
	var direction = Input.get_axis("ui_left", "ui_right")
	if direction:
		velocity.x = direction * SPEED
	else:
		velocity.x = move_toward(velocity.x, 0, SPEED)

	move_and_slide()
```

How do I know what keys “ui_left” and “ui_right” are mapped to?

According to the same Godot input map (screenshot), references to “ui_left” and “ui_right” are mapped to the left and right arrow keys on the keyboard respectively. Note: a Joypad Button device (a sort of handheld multi-button game controller) could also be used (i.e. its buttons are mapped to “ui_left” and “ui_right” too.



What does “is_on_floor() do?

```
extends CharacterBody2D
const SPEED = 130.0
const JUMP_VELOCITY = -300.0

# Get the gravity from the project settings to be synced with RigidBody nodes.
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
	# Add the gravity.
	if not is_on_floor():
		velocity.y += gravity * delta

	# Handle jump.
	if Input.is_action_just_pressed("ui_accept") and is_on_floor():
		velocity.y = JUMP_VELOCITY

	# Get the input direction and handle the movement/deceleration.
	# As good practice, you should replace UI actions with custom gameplay actions.
	var direction = Input.get_axis("ui_left", "ui_right")
	if direction:
		velocity.x = direction * SPEED
	else:
		velocity.x = move_toward(velocity.x, 0, SPEED)

	move_and_slide()
```

What does Godot “is_on_floor()” function do?

According to Godot document below:

https://docs.godotengine.org/en/stable/classes/class_characterbody2d.html#class-characterbody2d-method-is-on-floor

The *is_on_floor()* function “Returns true if the body collided only with the floor on the last call of move_and_slide. Otherwise, returns false. The up_direction and floor_max_angle are used to determine whether a surface is ‘floor’ or not.”

Why did I call the *is_on_floor()* function in my game when the player’s character is jumping? Without that function, my game would allow a player to jump while in mid-air! I will provide a demonstration with the *is_on_floor()* function call removed from my work-in-progress game to show you how it impacts the game play.

Live demo of a work-in-progress 2D game.

Advice on developing your first game

The difficulties you will experience in making your first Godot game:

- Navigating the user-interface in Godot (yes, it is complicated).
- Learning the Godot application programmer's interface (API) to achieve the sorts of actions you want the game to achieve.

To alleviate the steep learning curve, I suggest:

- Don't speed-run Godot's Dodge the Creeps tutorial. There are many details to follow and understand. Your attention to details matters!
- There are numerous informative Youtube videos on game development in Godot. Watch them. You can pause the video as you watch.
- Everyone learns at a different pace.
- Ask me questions during office hours. I'm here to help!

Excellent Godot tutorial

How to make a Video Game - Godot Beginner Tutorial

https://www.youtube.com/watch?v=LOhfqjmasi0

Art assets for game development

Godot Asset Library

https://godotengine.org/asset-library/asset