Casters & Coders

Advisor/Client: Mat Wymore
By: sddec23-13

Branden Butler
Wenqin Wu
Edward Dao

Max Bromet
Theng Wei Lwe
Brennan Seymour

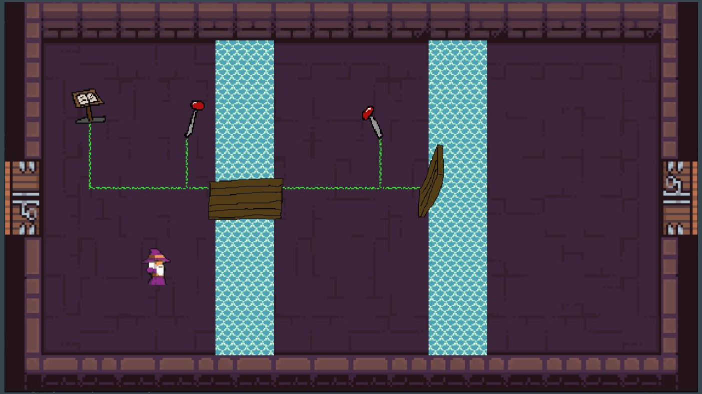
Motivation and goal

Some people might want to learn basic programming with little to no background knowledge, so we set out to create a high-fantasy puzzle game which:

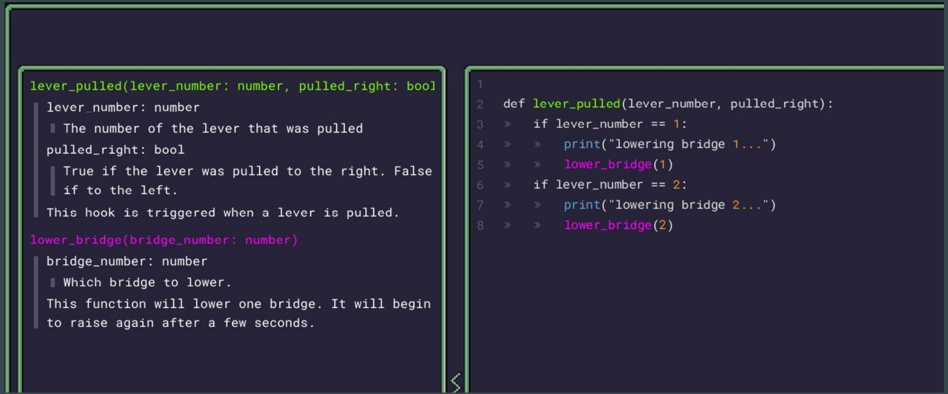
- Has coding as a main gameplay mechanic
- Teaches basic programming concepts
- Requires little to no background knowledge
- Is fun to play

Overview

- 2D, Top-Down, sprite-based video game
- User control a sprite character by using their keyboards
- There are rooms with puzzles that the player can solve by writing python scripts
- An editor screen to write scripts
- Scripts will control elements of the environment



Game Environment



```
lever_pulled(lever_number: number, pulled_right: bool
  lever_number: number
    The number of the lever that was pulled
  pulled_right: bool
    True if the lever was pulled to the right. False
    if to the left.
  This hook is triggered when a lever is pulled.
lower_bridge(bridge_number: number)
  bridge_number: number
    Which bridge to lower.
  This function will lower one bridge. It will begin
  to raise again after a few seconds.
```

```python

def lever_pulled(lever_number, pulled_right):
    if lever_number == 1:
        print("lowering bridge 1...")
        lower_bridge(1)
    if lever_number == 2:
        print("lowering bridge 2...")
        lower_bridge(2)
```

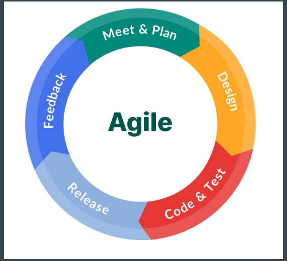
Text Editor

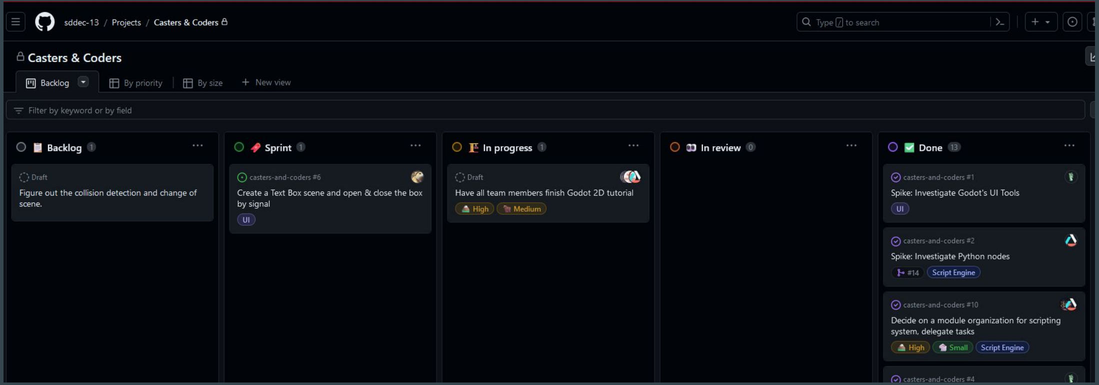
Who will benefit from our project?

- ★ Students
- ★ Teachers
  - Educational Institutions
  - Technology Industry

Project Management

- We adhere to the agile project management style
- With a video game there were a lot of testing and debugging
- Requirements give us a framework for setting long and short term goals
- Additional features can be dropped and added throughout the project
- GitHub, User Stories, Sprint Boards, Spring retrospective





sddec-13 / Projects / Casters & Coders

Casters & Coders

Backlog | By priority | By size | New view

Filter by keyword or by field

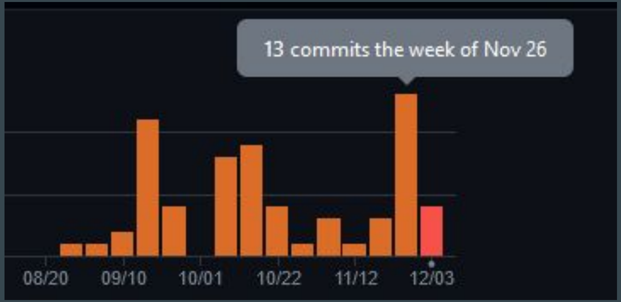
Backlog 1
- Draft
  Figure out the collision detection and change of scene.

Sprint 1
- casters-and-coders #6
  Create a Text Box scene and open & close the box by signal
  UI

In progress 1
- Draft
  Have all team members finish Godot 2D tutorial
  High | Medium

In review 0

Done 13
- casters-and-coders #1
  Spike: Investigate Godot's UI Tools
  UI
- casters-and-coders #2
  Spike: Investigate Python nodes
  #14 | Script Engine
- casters-and-coders #10
  Decide on a module organization for scripting system, delegate tasks
  High | Small | Script Engine
- casters-and-coders #4



Requirements & Constraints

Functional:

- The player must write scripts to perform actions
- Scripts will interact with puzzles through a predetermined API
- The editor must allow the player to write scripts, and display available API elements

Resource:

- The game should run at 60 fps on minimum specs
- Minimum spec constraints are: 4GB RAM, AMD A10-5800k CPU w/ Radeon HD, 10GB HDD space

UI:

- The game should be highly accessible
- The game should be fully functional using only a keyboard
- The game should use high contrast colors for the code editor

Qualitative:

- The game should have engaging puzzles
- The game shouldn't be frustrating

Economic:

- The game should be free and open source
- Only free and self-made assets will be used

Project Milestones

Engine and Scripting System

- Engine and Scripting Language locked in
- Script API implemented and documented

Story, Environment Design, and Asset Creation

- Completed map design and designs of characters

Puzzle Design

- Learning Pathway completed
- Introductory, Exploratory, and Challenging Puzzles designed

Mechanics of the Game and Level Implementation

- Fully functional player character
- Interactable objects

User Interfaces

- In-game IDE for user input with syntax highlighting

Technical Details

SCRIPTING LANGUAGE & GAME ENGINE

Python

- Easy to use and embed
- Very common language in real world
- Common beginner language
- Dynamic / Duck typing
- Not prototype-based

Godot

- Supports many game logic languages through bindings
- Result - easy to embed scripting languages
- Node system lends itself well to our needs
- Use of signals - Straightforward, easy to implement

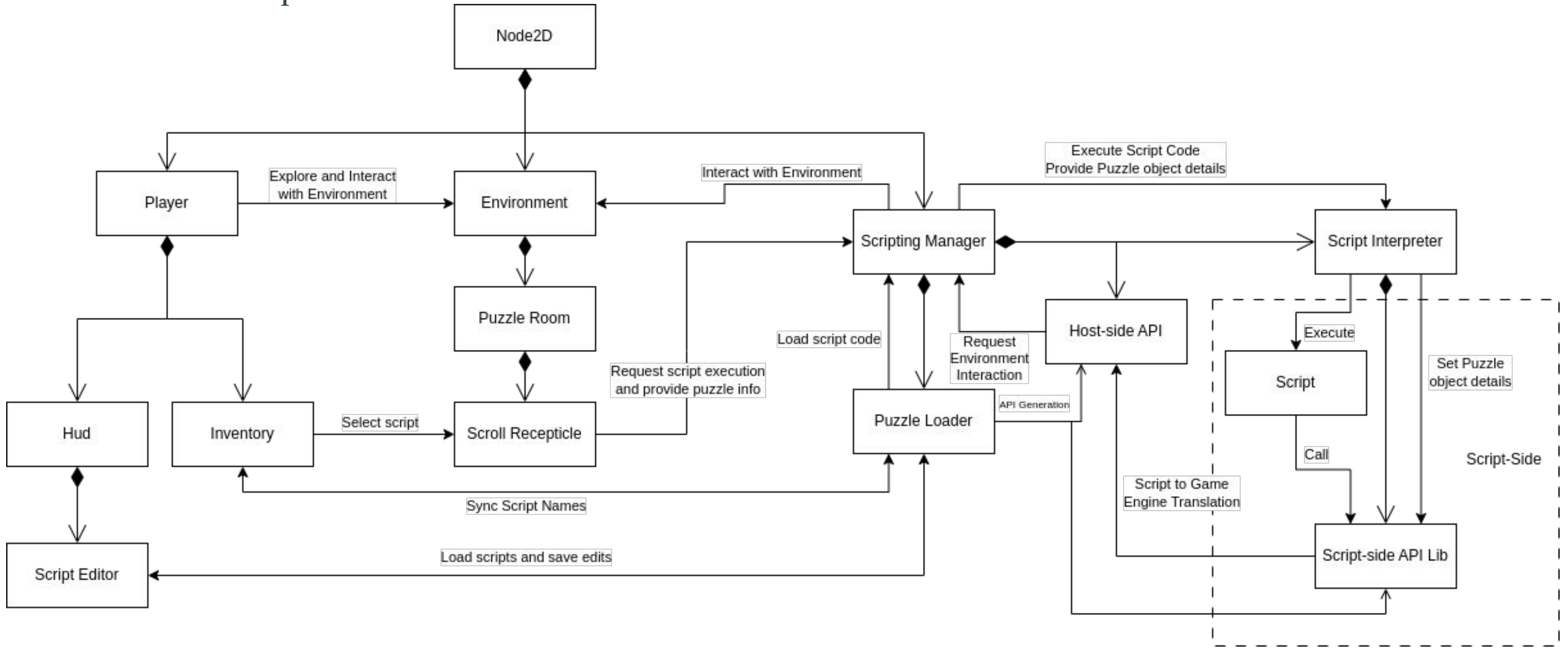
Puzzle Design

PUZZLES

- Major concept in this game
- Users will write scripts that directly affect objects or states in the game.
- Scripts directly control objects in the environment
- Success is measured in traversal, or other world-interaction
    - Traversal is an intuitive way to represent player success and is often used to represent player success.
    - Engaging, interesting, players want to do more

System Design

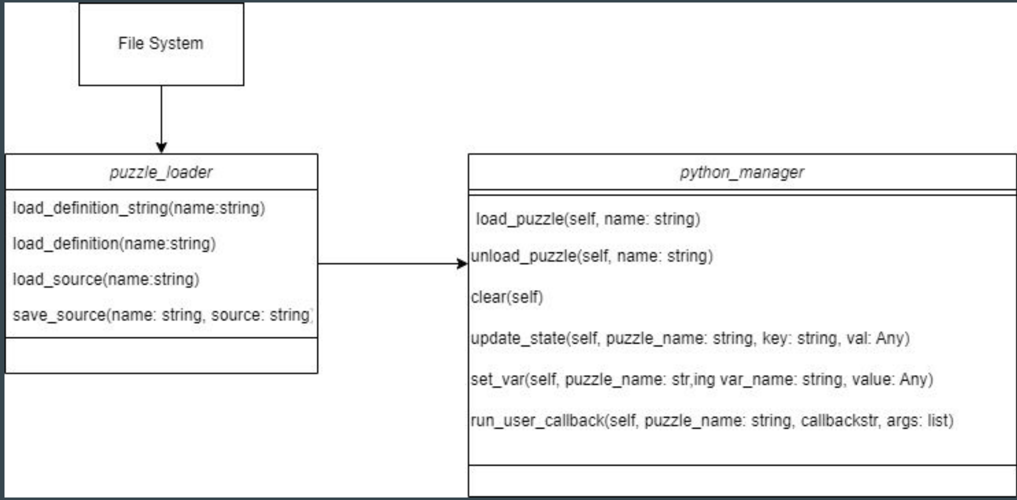
Godot Node Graph



Node2D
Player
Explore and Interact with Environment
Environment
Interact with Environment
Execute Script Code Provide Puzzle object details
Scripting Manager
Script Interpreter
Puzzle Room
Load script code
Request Environment Interaction
Host-side API
Execute
Script
Set Puzzle object details
Request script execution and provide puzzle info
Hud
Inventory
Select script
Scroll Recepticle
Puzzle Loader
API Generation
Call
Script-Side
Sync Script Names
Script to Game Engine Translation
Script Editor
Load scripts and save edits
Script-side API Lib

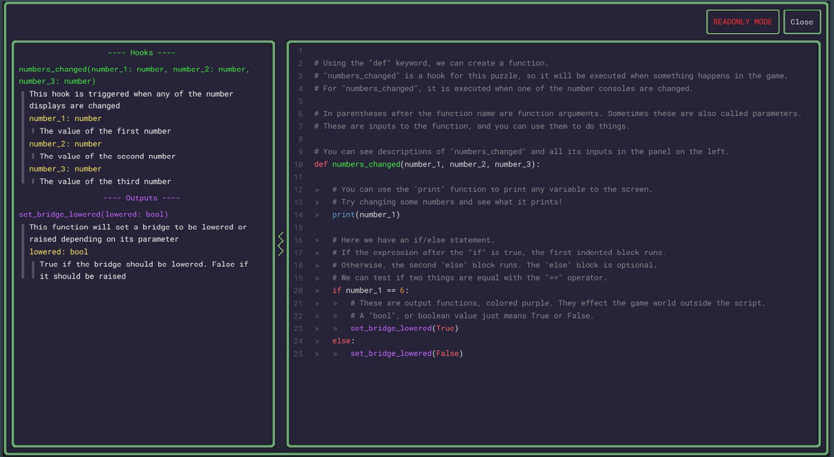
Scripting System Team

- Scripting system architecture enables real time execution and editing of user scripts
- Makes heavy use of Python reflection, closures, and dynamically generated functions
- System is exposed to Godot through the Godot Python bindings
- Puzzle is defined through JSON files
  - Defines the set of inputs and outputs
  - Functions are automatically generated
  - Print functions overridden
- User script source is loaded and compiled
  - Python 3 compiles source to bytecode
  - Compilation checks for syntax errors


File System
puzzle_loader
load_definition_string(name:string)
load_definition(name:string)
load_source(name:string)
save_source(name: string, source: string
python_manager
load_puzzle(self, name: string)
unload_puzzle(self, name: string)
clear(self)
update_state(self, puzzle_name: string, key: string, val: Any)
set_var(self, puzzle_name: str,ing var_name: string, value: Any)
run_user_callback(self, puzzle_name: string, callbackstr, args: list)

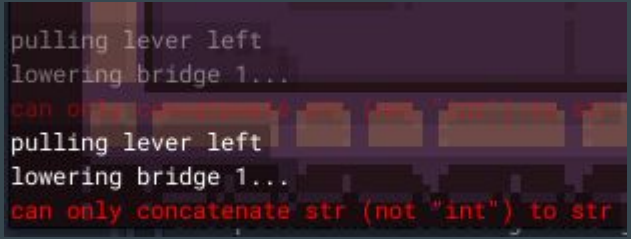
User Interface Team

- Syntax highlighting
- Generated API docs
- Readonly mode
- Prefilled scripts
- On-screen log for stdout & exceptions

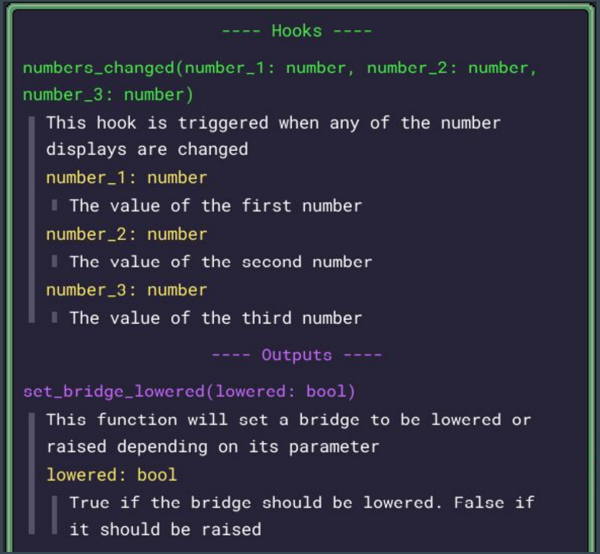


```
---- Hooks ----
numbers_changed(number_1: number, number_2: number,
number_3: number)
  This hook is triggered when any of the number
  displays are changed
  number_1: number
    The value of the first number
  number_2: number
    The value of the second number
  number_3: number
    The value of the third number
---- Outputs ----
set_bridge_lowered(lowered: bool)
  This function will set a bridge to be lowered or
  raised depending on its parameter
  lowered: bool
    True if the bridge should be lowered. False if
    it should be raised
```

```python

# Using the "def" keyword, we can create a function.
# "numbers_changed" is a hook for this puzzle, so it will be executed when something happens in the game.
# For "numbers_changed", it is executed when one of the number consoles are changed.

# In parentheses after the function name are function arguments. Sometimes these are also called parameters.
# These are inputs to the function, and you can use them to do things.

# You can see descriptions of "numbers_changed" and all its inputs in the panel on the left.
def numbers_changed(number_1, number_2, number_3):

    # You can use the "print" function to print any variable to the screen.
    # Try changing some numbers and see what it prints!
    print(number_1)

    # Here we have an if/else statement.
    # If the expression after the "if" is true, the first indented block runs.
    # Otherwise, the second "else" block runs. The "else" block is optional.
    # We can test if two things are equal with the "==" operator.
    if number_1 == 6:
        # These are output functions, colored purple. They effect the game world outside the script.
        # A "bool", or boolean value just means True or False.
        set_bridge_lowered(True)
    else:
        set_bridge_lowered(False)
```



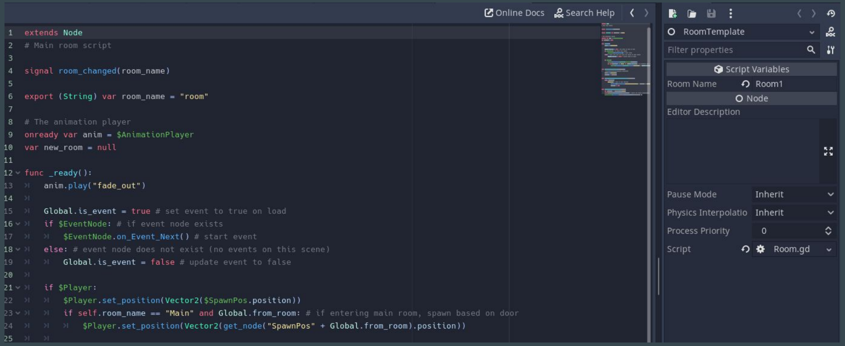
```
pulling lever left
lowering bridge 1...
pulling lever left
lowering bridge 1...
can only concatenate str (not "int") to str
```



```
---- Hooks ----
numbers_changed(number_1: number, number_2: number,
number_3: number)
  This hook is triggered when any of the number
  displays are changed
  number_1: number
    The value of the first number
  number_2: number
    The value of the second number
  number_3: number
    The value of the third number
---- Outputs ----
set_bridge_lowered(lowered: bool)
  This function will set a bridge to be lowered or
  raised depending on its parameter
  lowered: bool
    True if the bridge should be lowered. False if
    it should be raised
```

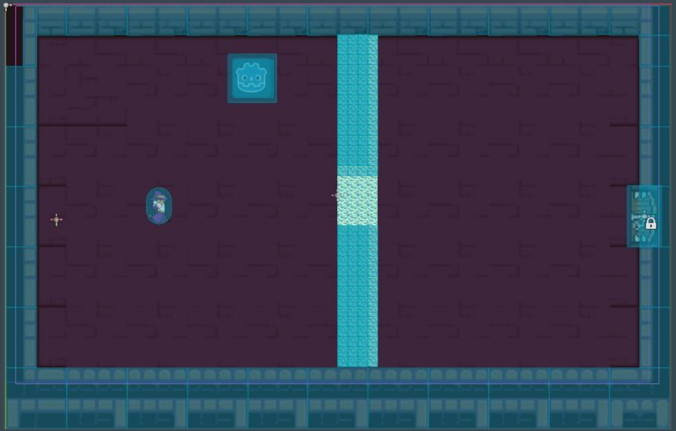
Game Logic Team

- Designed the tilemap, character, objects, object interactions, collision masks & layers
- Implemented the Character movements and overall main dungeon scenes
- Implemented room traversal logic
- Implemented dialog boxes
- Set the base groundwork for having the work from the UI team and Python Scripting System team to be easily integrated



```
extends Node
# Main room script

signal room_changed(room_name)

export (String) var room_name = "room"

# The animation player
onready var anim = $AnimationPlayer
var new_room = null

func _ready():
	anim.play("fade_out")
	
	Global.is_event = true # set event to true on load
	if $EventNode: # if event node exists
		$EventNode.on_Event_Next() # start event
	else: # event node does not exist (no events on this scene)
		Global.is_event = false # update event to false
	
	if $Player:
		$Player.set_position(Vector2($SpawnPos.position))
		if self.room_name == "Main" and Global.from_room: # if entering main room, spawn based on door
			$Player.set_position(Vector2(get_node("SpawnPos" + Global.from_room).position))
		
```

Room Name: Room1
Pause Mode: Inherit
Physics Interpolatio: Inherit
Process Priority: 0
Script: Room.gd



Retrospective — What Has Been Accomplished

- Successfully implemented the game with proof-of-concept round trip
- Fulfilled client's requirements
- We separated project into 3 sub-branches and each one of them did a good job.
  - Core game logic implemented
  - Interactive UI
  - Working Python Script Manager
- Have an organized repository that is easily manageable

Retrospective — Next Steps and Game Potential

- Sandboxing/Coding playground
- Integrate a storyline along the way
- Save game progress
- More challenging puzzles (LEET code level)
- Have completed puzzles be levels that can be revisited but with twists (extra difficulty)

Short Game Demo

•••

Q&A

●●●