Spring 2023

# From Filth it Rises presentation

Griffin Hartz

# From Filth It Rises

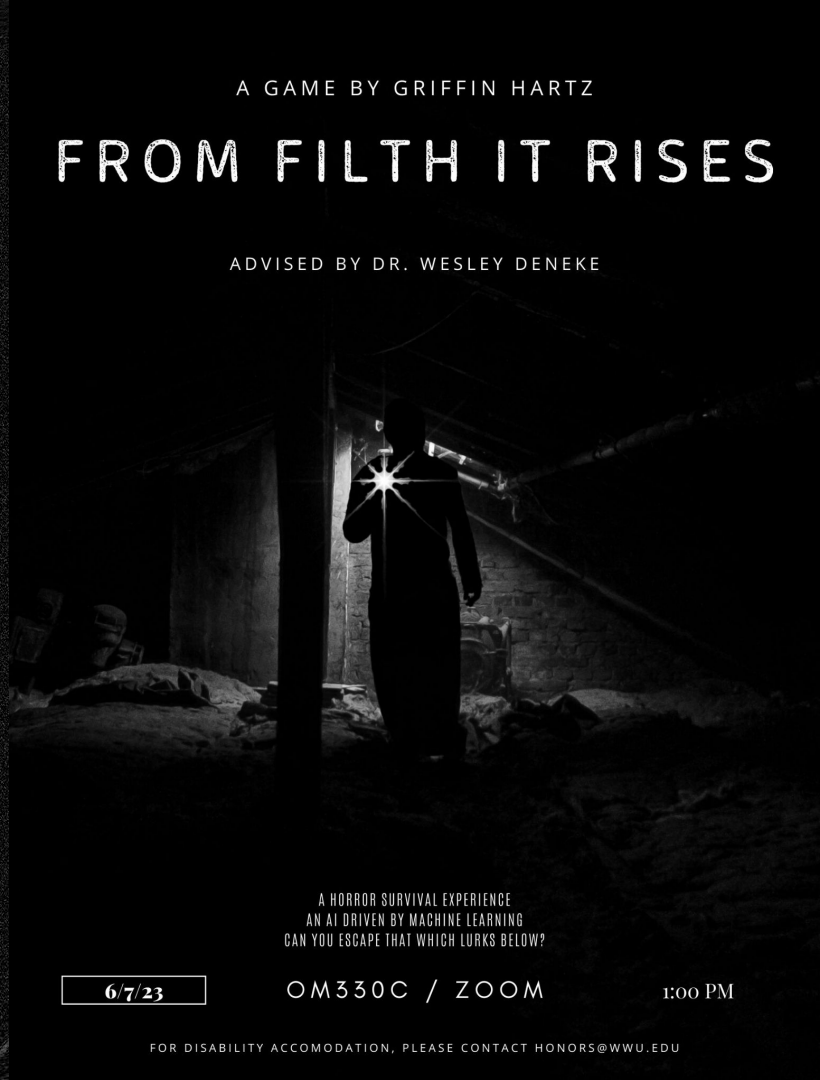
Project by Griffin Hartz

Advised by Wesley Deneke

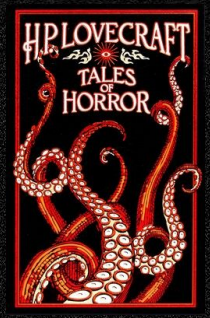
# Overview

- Video game made in Unity Engine over Spring Quarter
- Genre: Survival Horror
- AI for enemy developed using Unity's Machine Learning Agents package
- I had no background in ML prior to project

unity



# My Background





- Computer Science Major
- Done research with Dr.Deneke developing a virtual world since fall 2021
  - Currently developing it as a therapy platform
  - Have worked on Virtual Reality, Networking, User Abilities & Tools, Database, Tutorials, Environments
  - Worked on it last summer via the Elwha award scholarship from WWU
- President of WWU Game Design Club since fall 2022
  - Officer since fall 2021
- Took Dr.Deneke's Game Programming class fall 2021
  - Made a dark fantasy stealth game called *'Tainted Whisper'*
- Fan of:
  - Tabletop & video games
  - Metal music
  - Scifi/fantasy/horror books & movies



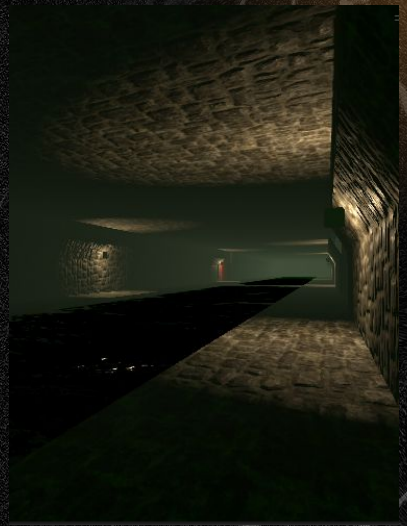
# Game Concept / Motivation

- What is a survival horror game?
  - Player is generally in a confined space they must escape.
  - Limited power/control.
    - Often need to subvert foes rather than fight.
- What makes survival horror interesting?
  - Player needs a different approach than action or FPS games.
    - Alternatives to direct confrontation.
  - Exploration of fear in an immersive context.
- What makes these games challenging?
  - Scarce resources.
  - Use of environment.
  - Puzzle elements.



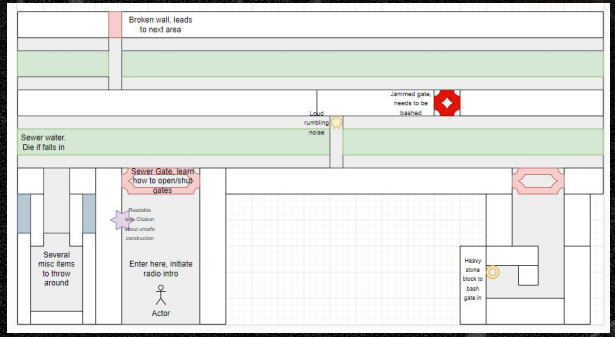
# Game Concept cont.

- Player is sent to investigate a sewer beneath Cologne, Germany.
- A previous employee had been sent down earlier.
  - Did not return, is not answering radio.
- Player goes down and finds a collapsed wall.
- Leads to a buried undercity.
  - Accessible through a gate which closes behind them.
  - Radio does not work in the city.
- Coworker is quickly found, slaughtered.
- There is a mutant monster prowling the undercity.
  - Hunts the player.
- Player has to find a way out of the city, or a way to contact help.



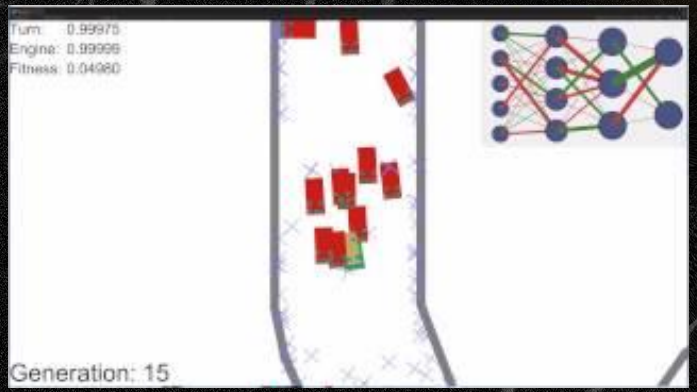
# Early design and development

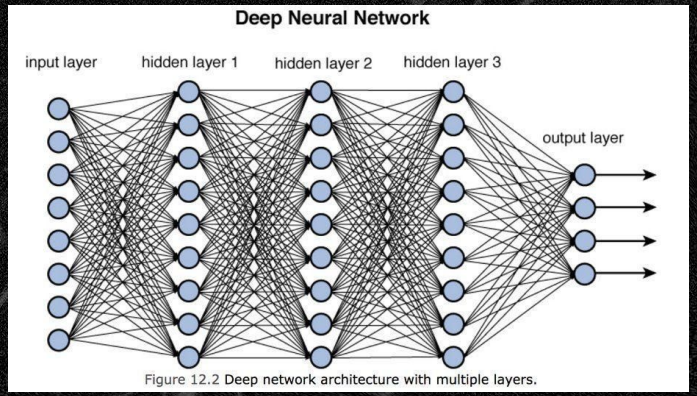
- Level design sketching based on expected player movement
- Interactions, some inspired by other works
  - Readable items for info about setting, hints
  - Throwable objects
    - Distractions
    - Breaking obstacles
  - Obstacles such as jammed gates
- Mutant is to pursue ‘traces’ the player drops
- WASD movement and cursor based looking
- Design document




# What is a Neural Network?

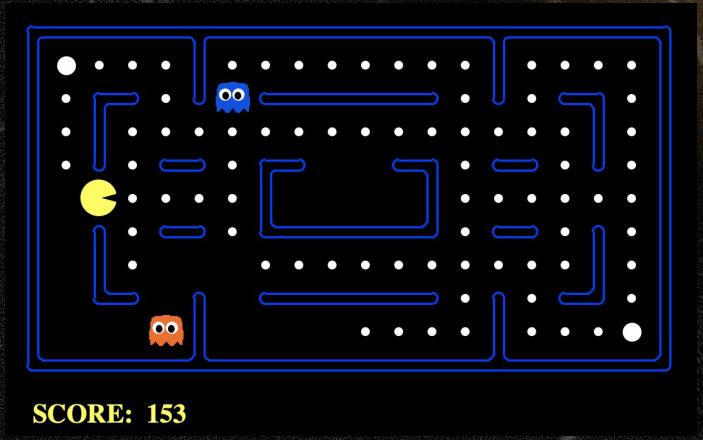
- Layers of nodes/neurons.
- Input layer, x amount of hidden layers, output layer.
- Input layer receives some form of data, usually numbers.
- Neurons receive input from previous layer.
  - Perform a mathematical operation.
  - Apply activation function.
    - Activation function stimulates non-linearity, allowing complexity.
- Output layer returns predictions from hidden layers.

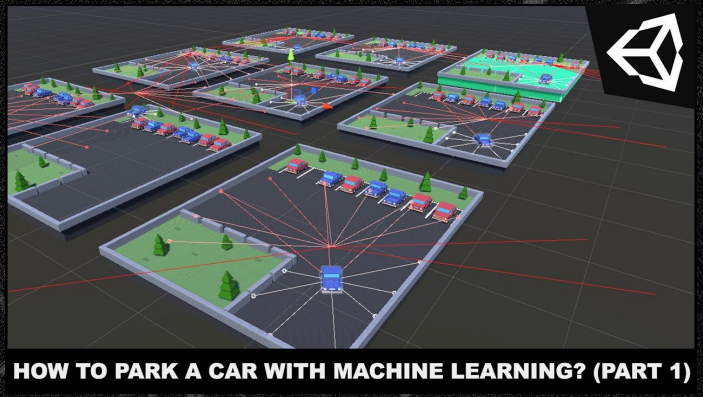


# Neural Network Cont.

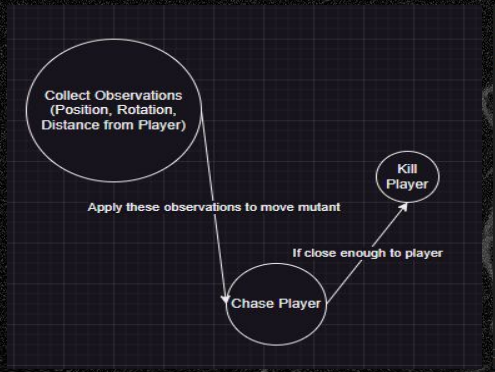
- Subject the NN to a training scenario which will produce the input data
- During training, NN adjusts internal parameters (weights & biases)
  - This is based on input data & desired output
- Unity's MLAgents:
  - Uses a reward-punishment system
- Why use ML as opposed to static AI?
  - Flexibility



# Integrating Machine Learning

- Training runs via ‘episodes’ that are a given amount of ‘steps.’
- CollectObservations & OnActionRecieved
  - Input and output, respectively
- Output Visible in animator
  - Using this to drive animation, movement

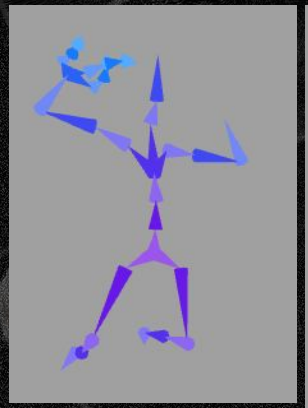


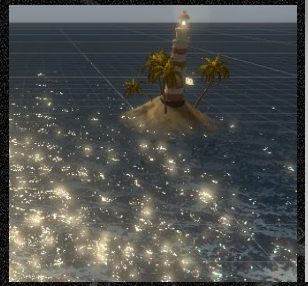

```
/// <summary>
/// Collect vector observations from the environment
/// </summary>
/// <param name="sensor">The vector sensor</param>
0 references
public override void CollectObservations(VectorSensor sensor)
{
    //Debug.Log("Collecting observations");

    // If nearestPlayer is null, observe an empty array and return early
    if (nearestPlayer == null)
    {
        Debug.Log("return early observation, player null");
        sensor.AddObservation(new float[10]);
        return;
    }

    // Observe the agent's local rotation (4 observations)
    sensor.AddObservation(transform.localRotation.normalized);

    // Get a vector from the claw to the nearest player
    Vector3 toPlayer = nearestPlayer.PlayerCenterPosition - claw.position;
    //Debug.Log("To player: " + toPlayer);

    // Observe a normalized vector pointing to the nearest player (3 observations)
    sensor.AddObservation(toPlayer.normalized);

    // Observe a dot product that indicates whether the claw is in front of the player (1 observation)
    // (+1 means that the claw is directly in front of the player, -1 means directly behind)
    sensor.AddObservation(Vector3.Dot(toPlayer.normalized, -nearestPlayer.PlayerUpVector.normalized));

    // Observe a dot product that indicates whether the beak is pointing toward the player (1 observation)
    // (+1 means that the beak is pointing directly at the player, -1 means directly away)
    sensor.AddObservation(Vector3.Dot(claw.forward.normalized, -nearestPlayer.PlayerUpVector.normalized));

    // Observe the relative distance from the claw to the player (1 observation)
    sensor.AddObservation(toPlayer.magnitude);
}
```

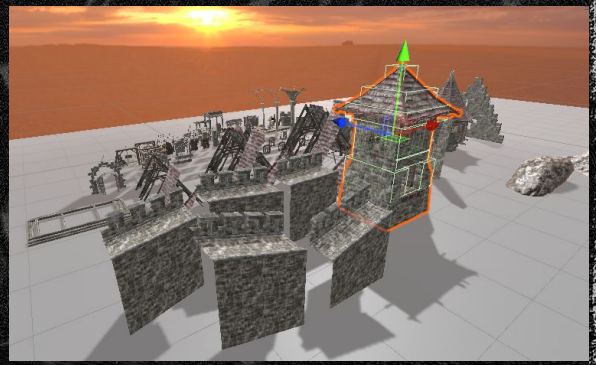
# Training Demo

# Asset collection

- I had several unity assets from previous projects available to me (some were free)
- These include:
  - Castle & mine visual 3D assets and materials
  - Water shaders
  - Footstep, fantasy & horror sounds
  - Unity's first-person demo project
- Got mutant model & animations from Mixamo.com



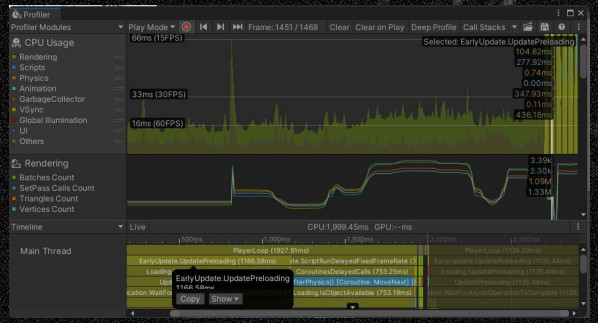


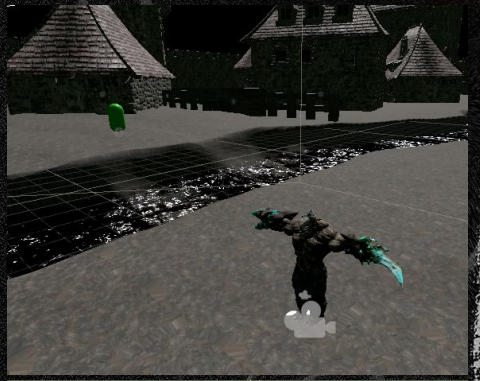


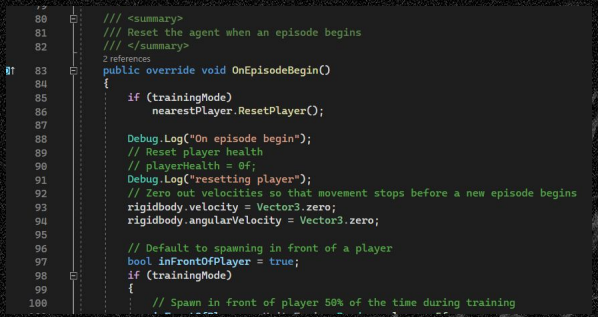
# Gameplay Demo

# Development Process

- Learn MLAgents
  - Environment setup
  - Basic training scenario
  - Avatar rigging & animation
- Level design
  - Whiteboxing
- Adding player
  - Basic actions
- Interactables
- Curating visuals



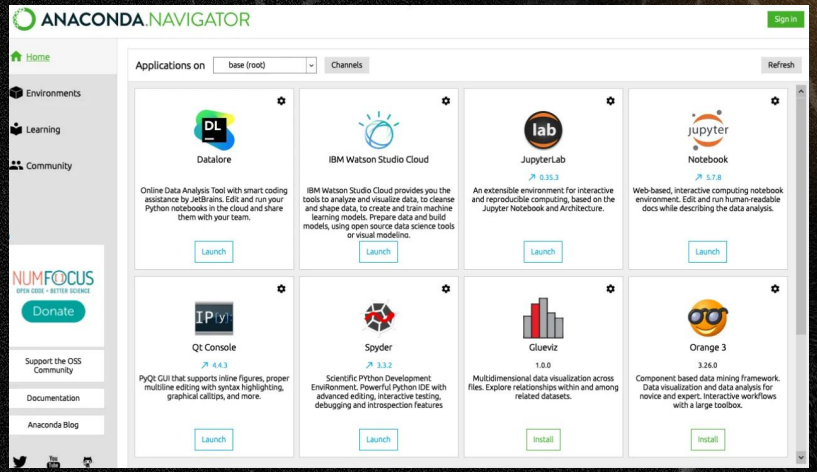





```
80      /// <summary>
81      /// Reset the agent when an episode begins
82      /// </summary>
        2 references
83      public override void OnEpisodeBegin()
84      {
85          if (trainingMode)
86              nearestPlayer.ResetPlayer();
87
88          Debug.Log("On episode begin");
89          // Reset player health
90          // playerHealth = 0f;
91          Debug.Log("resetting player");
92          // Zero out velocities so that movement stops before a new episode begins
93          rigidbody.velocity = Vector3.zero;
94          rigidbody.angularVelocity = Vector3.zero;
95
96          // Default to spawning in front of a player
97          bool inFrontOfPlayer = true;
98          if (trainingMode)
99          {
100             // Spawn in front of player 50% of the time during training
```

# Challenges

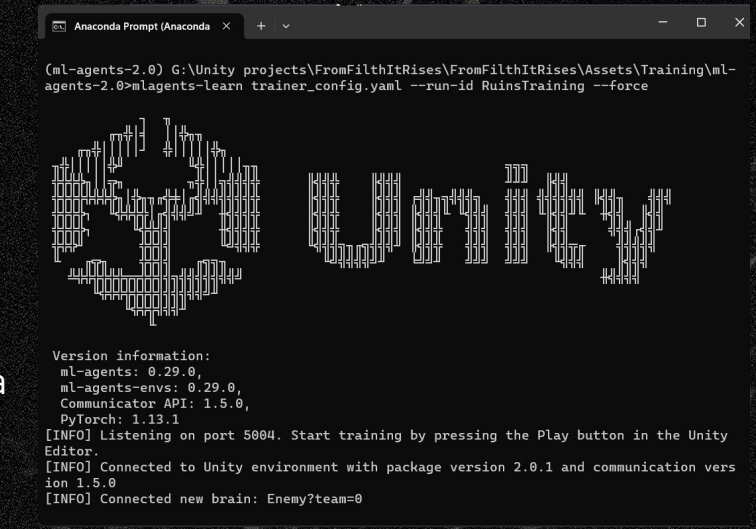
- Git (version control) with Unity
- Anaconda environment setup
- Learning MLAgents was time consuming
- Animation with ML was difficult
- Extending default interactions
- Managing all elements without team




```
  return (tensor.T * masks).sum() / torch.clamp(
[INFO] Enemy. Step: 20000. Time Elapsed: 395.878 s. Mean Reward: 3.738.
 Std of Reward: 4.810. Training.
[INFO] Enemy. Step: 30000. Time Elapsed: 558.735 s. Mean Reward: 3.083.
 Std of Reward: 6.307. Training.
[INFO] Enemy. Step: 40000. Time Elapsed: 740.971 s. Mean Reward: 2.277.
 Std of Reward: 5.756. Training.
[INFO] Enemy. Step: 50000. Time Elapsed: 908.094 s. Mean Reward: 2.425.
 Std of Reward: 5.927. Training.
```

# Current State & Next Steps

- Game still needs a handful of functionality:
  - Improved UI & Controls options
  - Additional interactable items
  - Sound not implemented
- AI needs additional training & tweaking
  - Still does an annoying amount of random wandering
  - Will eventually do heuristic training with a live player
- Pickup items & some other objects use placeholder assets
- Readable text throughout the ruins



# Questions?