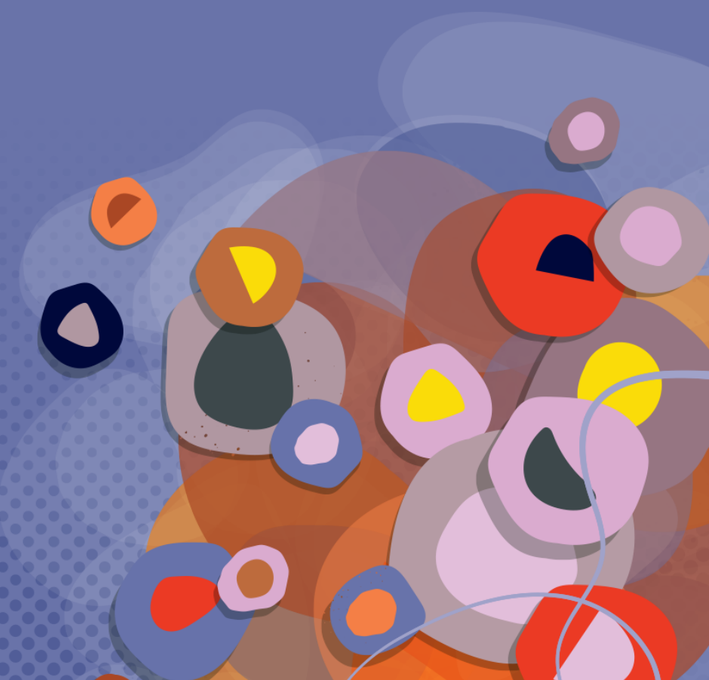
GDC March 20-24, 2023 San Francisco, CA

Bots at Work: Lessons on Deploying ML Bots in Your Game

Ricardo Sisnett, Lead Engineer @ modl.ai



#GDC23

Have you ever been here?

Have you ever been here?

AI Summit: The Dimensional Curse of AAA Game Balancing: Data-Efficient and Robust Reinforcement Learning

Edgar Handy (Machine Learning Engineer, SQUARE ENIX CO., LTD.)
Location: Room 2002, West Hall
Date: Monday, March 20
Time: 9:30 am - 10:30 am
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture
Vault Recording: Video

Have you ever been here?

AI Summit: The Dimensional Curse of AAA Game Balancing: Data-Efficient and Robust Reinforcement Learning

Edgar Handy (Machine Learning Engineer, SQUARE ENIX CO., LTD.)
Location: Room 2002, West Hall
Date: Monday, March 20
Time: 9:30 am - 10:30 am
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture
Vault Recording: Video

The Future of AI in Gaming (Presented by King)

Steve Collins (Chief Technology Officer , King)
Danny Lange (SVP of Artificial Intelligence and Machine Learning, Unity)
Paul Stephanouk (Head of Creative, Candy Crush , King)
Pete Wurman (Director, Sony AI America)
Mark DeLoura (Games and Education Technology Consultant, Level Up Games)
Location: Room 2000, West Hall
Date: Monday, March 20
Time: 1:20 pm - 2:40 pm
Pass Type: All Access Pass, Core Pass, Summits Pass, Expo Pass, Audio Pass, Independent Games Summit Pass
Topic: Production & Team Leadership, Programming
Format: Sponsored Session
Vault Recording: Video

Tweet Like Share Share

Have you ever been here?

AI Summit: The Dimensional Curse of AAA Game Balancing: Data-Efficient and Robust Reinforcement Learning

Edgar Handy (Machine Learning Engineer, SQUARE ENIX CO., LTD.)
Location: Room 2002, West Hall
Date: Monday, March 20
Time: 9:30 am - 10:30 am
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture
Vault Recording: Video

AI Summit: Advancing Content Creation with Generative AI

Stefano Corazza (VP, Head of Roblox Studio, Roblox)
Location: Room 2002, West Hall
Date: Monday, March 20
Time: 2:10 pm - 2:40 pm
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture
Vault Recording: Video

Tweet Like Share Share

Roblox envisions a future where anyone can create anything, anywhere. With over 60-

ed by King)

ependent

Only to realize…

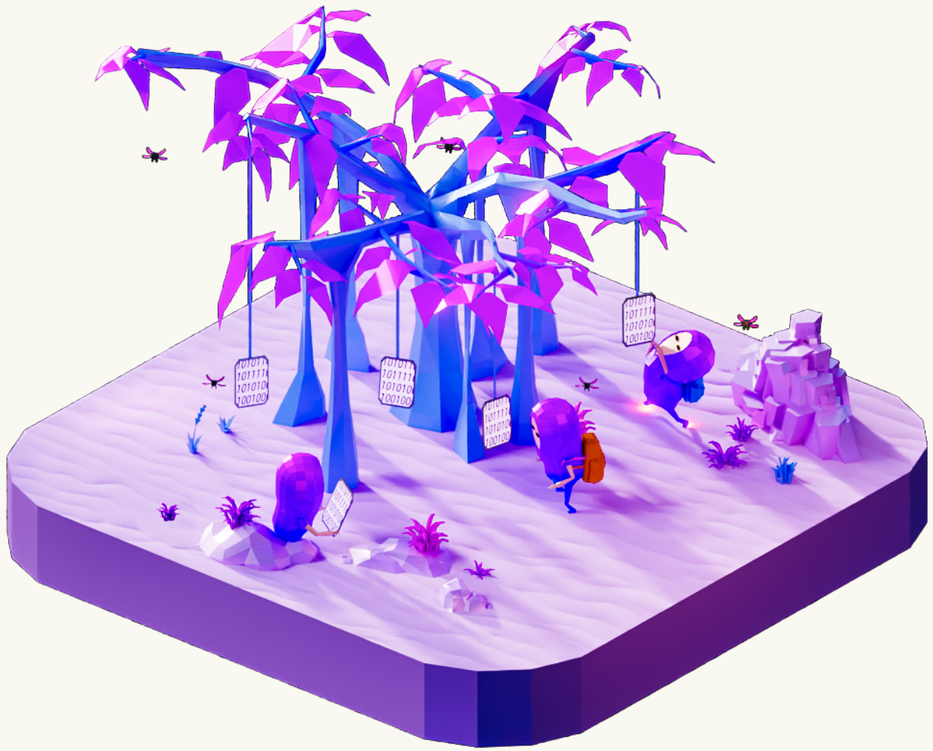
Only to realize…

Only to realize…

Why Are We All Here Today?

- Every major breakthrough in game technology has required a shift in how we architect games and game engines, adopting modern Machine Learning will not be the exception.

- We have a lot of space to grow, but multiple improvements can get us significant value with relatively low investment. However, those investments need to happen early in the development process.



Why Are You Listening to Me?

This is me!

Why Are You Listening to Me?

This is where I was born!

Why Are You Listening to Me?

I have loved games since they looked like this!

Why Are You Listening to Me?

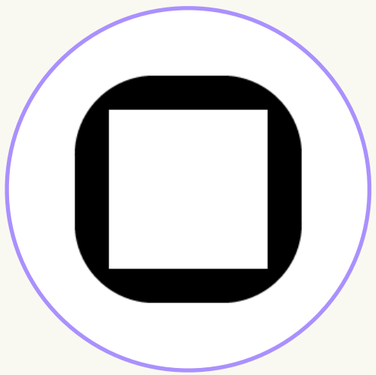
Built Enterprise Software

Why Are You Listening to Me?

Got to work on a cool game or 2

Why Are You Listening to Me?

Now I want to redefine how we make them!



Agenda

- Prologue - Definitions
  - High-Level overview of a few terms.
- Act 1 - Data
  - Why is data important and why do bots love it?
- Act 2 - Sensing & Acting
  - What do bots need to be able to exist in your world?
- Epilogue - A Bright Future

Prologue

Definitions

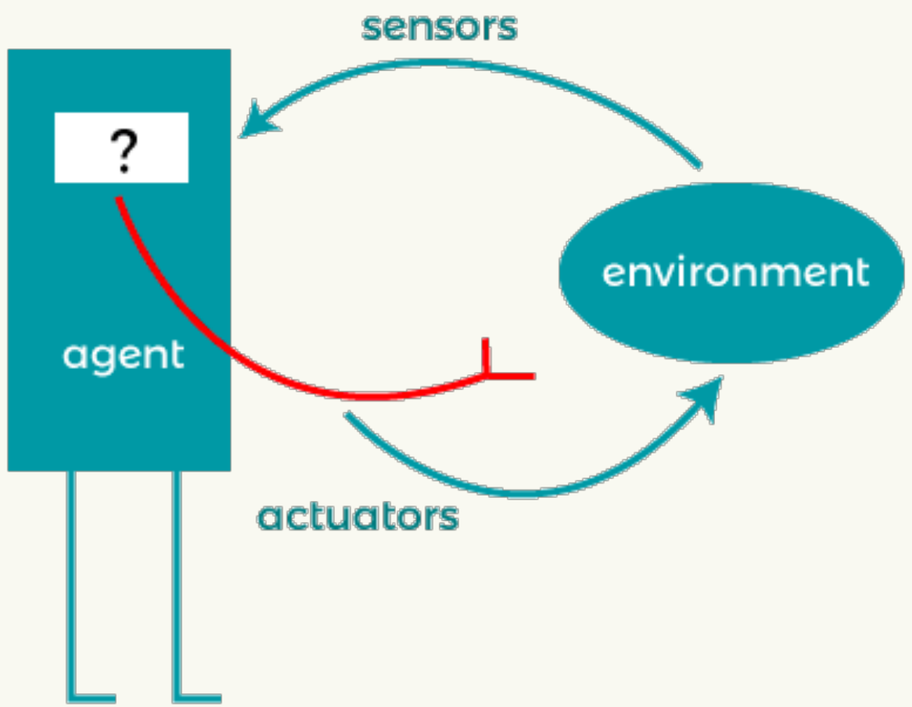
Machine Learning in a Nutshell

- Games can be summarized as:

```
F(Game State, Action) -> (New Game State, Reward)
```

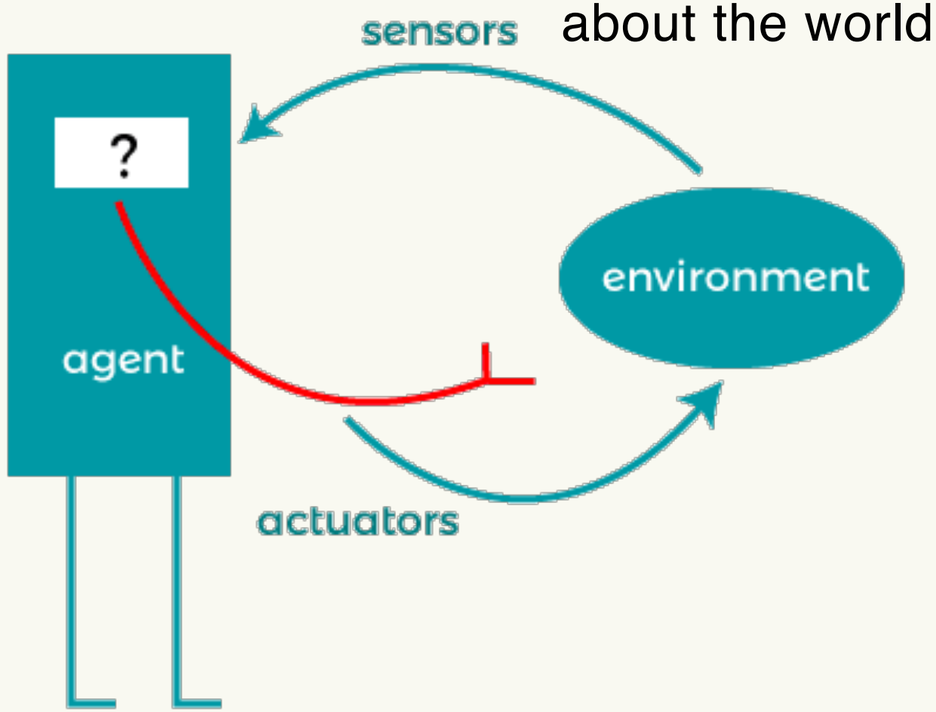
- ML Systems for games:
  - Given an observed game state, pick an action so that the reward is maximized.

Sense <> Think <> Act Loop



Sense <> Think <> Act Loop

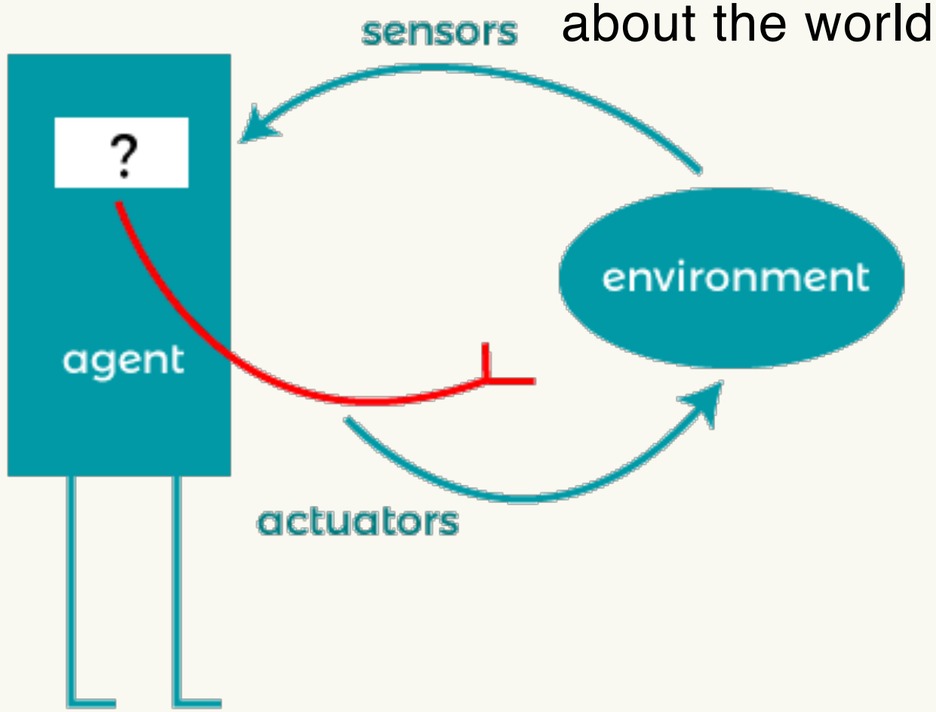
Sensors: Provide the agent with information about the world.



Sense <> Think <> Act Loop

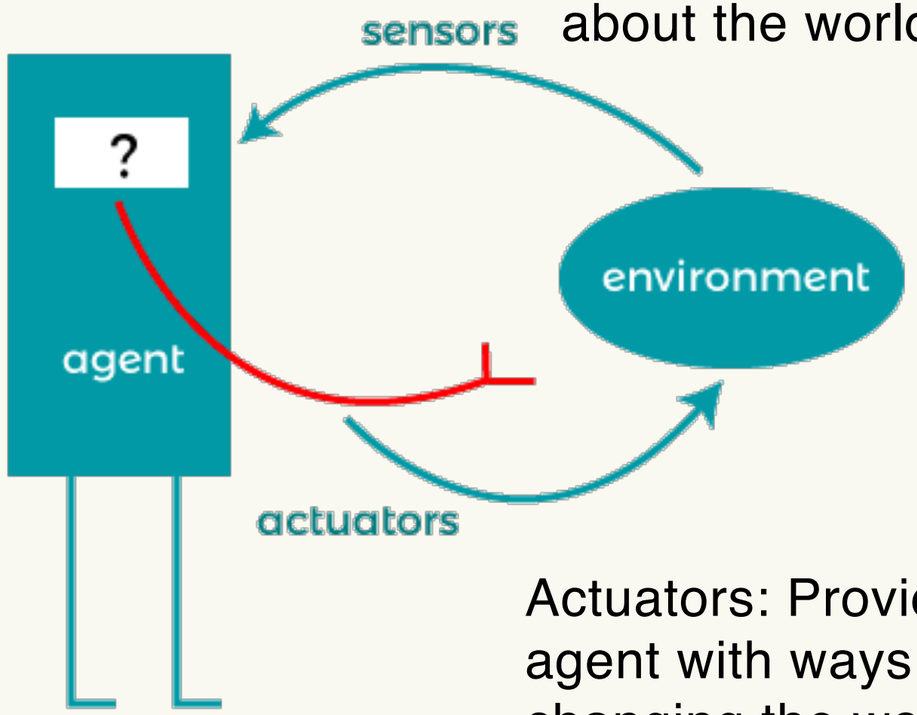
Think: The agent processes sensed information and provides an action or a set of actions.

Sensors: Provide the agent with information about the world.



Sense <> Think <> Act Loop

Think: The agent processes sensed information and provides an action or a set of actions.

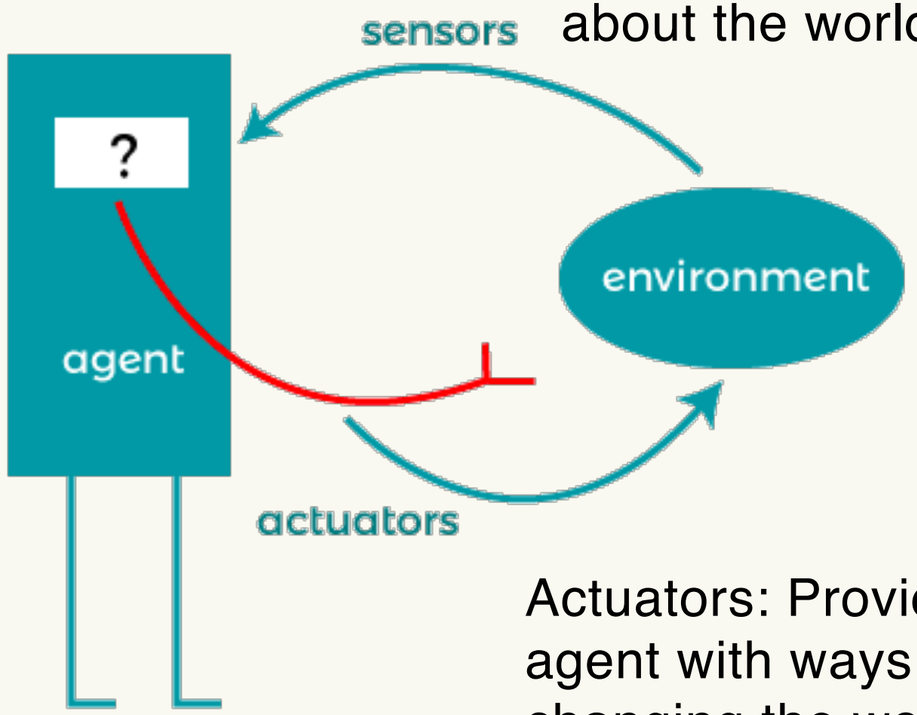


Sensors: Provide the agent with information about the world.

Actuators: Provide the agent with ways of changing the world.

Sense <> Think <> Act Loop

Think: The agent processes sensed information and provides an action or a set of actions.



Sensors: Provide the agent with information about the world.

Actuators: Provide the agent with ways of changing the world.

The Challenge:
These windows aren't well defined in modern games or game engines.

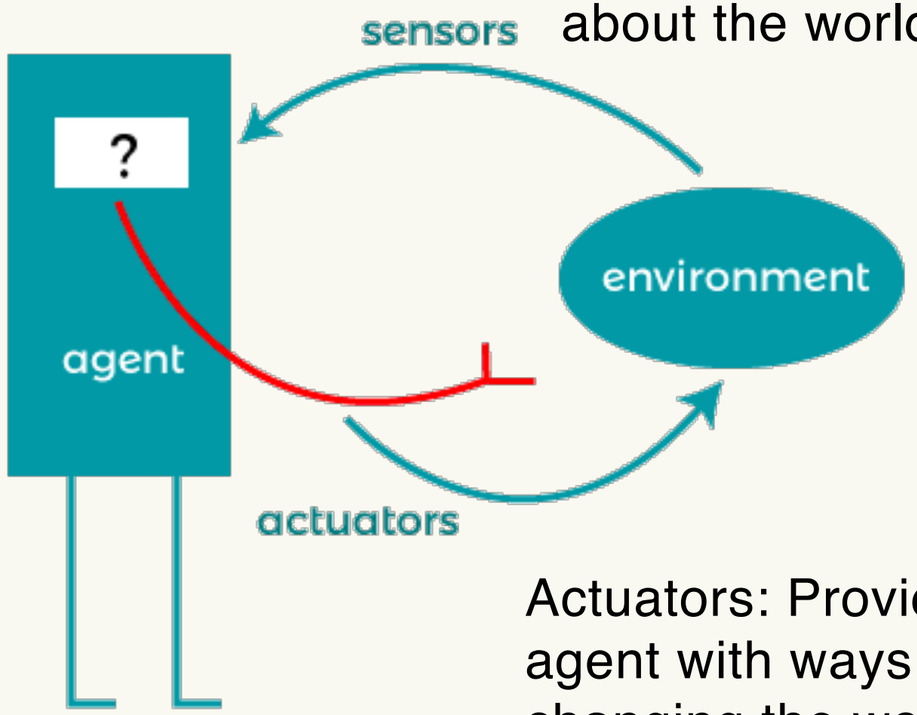
When do we sense?
What data is available?

When or how do we act?

Sense <> Think <> Act Loop

Think: The agent processes sensed information and provides an action or a set of actions.

The Other Challenge: Data is usually the defining factor of what your agent's brain can do.



Sensors: Provide the agent with information about the world.

Actuators: Provide the agent with ways of changing the world.

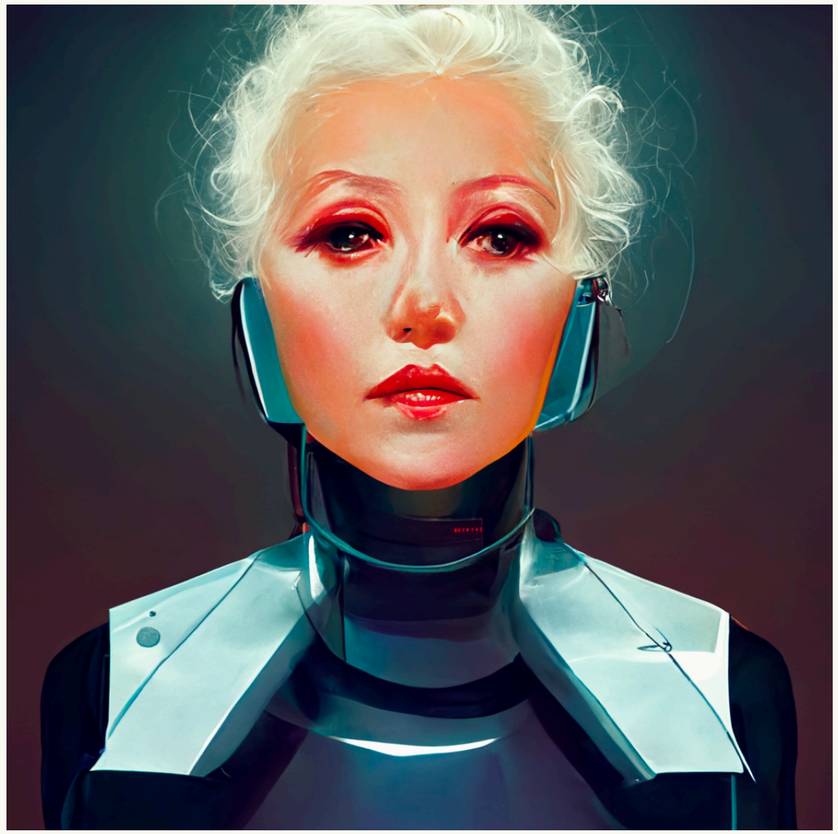
The Challenge:
These windows aren't well defined in modern games or game engines.

When do we sense?
What data is available?

When or how do we act?

What a Bot Wants… What a Bot Needs…

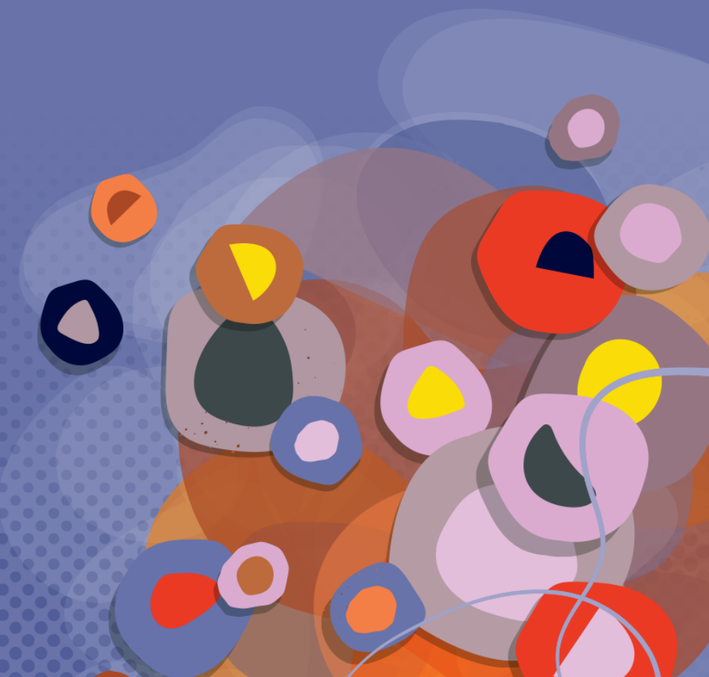
- Data:
  - Clean
  - Versioned
  - Easy to Experiment on
- Clear Interfaces:
  - To Manipulate your game
  - To Observe your game
  - Deterministically



“Christina AguilerAI”

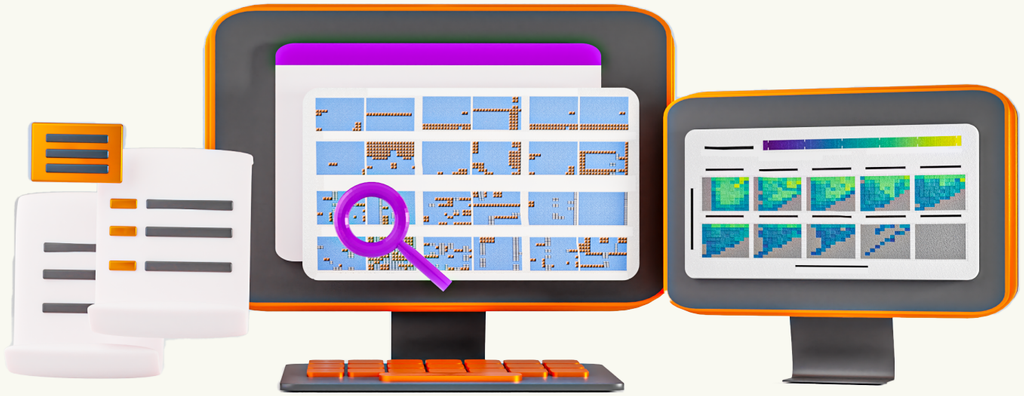
Act 1

Data



What’s Up With Data?

- ML systems are only as good as the data they have available.
  - Clean
    - What is the quality of your data captures?
  - Accessible
    - Is your data easy to access for people that want to experiment?
  - Consistent



Clean Data

- If you had access to the data you need how long would the script be to get that data ready for something like Tensorflow?
  - Detecting and Removing Outliers
  - Normalization of Values
  - Separation of concerns in Database Schemas - NoSQL? No Problem!

Accesible Data

- If you had someone come with a perfect model for your problem, how long would it take to give them access to experimental data to test said model?
  - Is your data accessible for your team?
  - Do you know the extent of the privacy concerns of the data you're experimenting with?
  - Is your Personal Identifiable Information (PII) separated from your experimental data?

Keeping Your PII in Place

```
{
  "session_id": 112233,
  "player_id": 332211,
  "started_on": "01/01/2020 01:00:00",
  "ended_on": "01/01/2020 01:10:00",
  "game_actions": { "..." }
}
```

Leaks PII when it isn't needed

```
{
  "session_id": 112233,
  "game_actions": { ... }
}
```

Keeps data concerns separated.
Requires a Session's Table

Consistent Data

- Is **Player A** doing something with **Item X** the same as **Player B** doing something with **Item X** 6 months later?
  - If not - Can you tell the difference?
- Add the patch version of your game in the data - PLEASE!
- Centralize your schemas through Protobuf or Flatbuffers can help.

Establishing a Data Collection Practice

- Bring in your data scientist as early as possible
  - Or consult with someone in a different studio
  - Or the community
  - Or contract one for a small period

Summary of Act 1

- Data will make or break your Machine Learning efforts and systems.
  - Think of it early, think of it often
  - Good data, is only as good as your org's ability to use it

- Cleaning your data has multiple facets:
  - Filtering, Removing Outliers and Normalizing Values
  - Centralizing your Database schemas can help with PII and the chaos of NoSQL databases.
  - Tackling these data architecture challenges becomes harder over time, bring someone early to the conversation.

Summary of Act 1

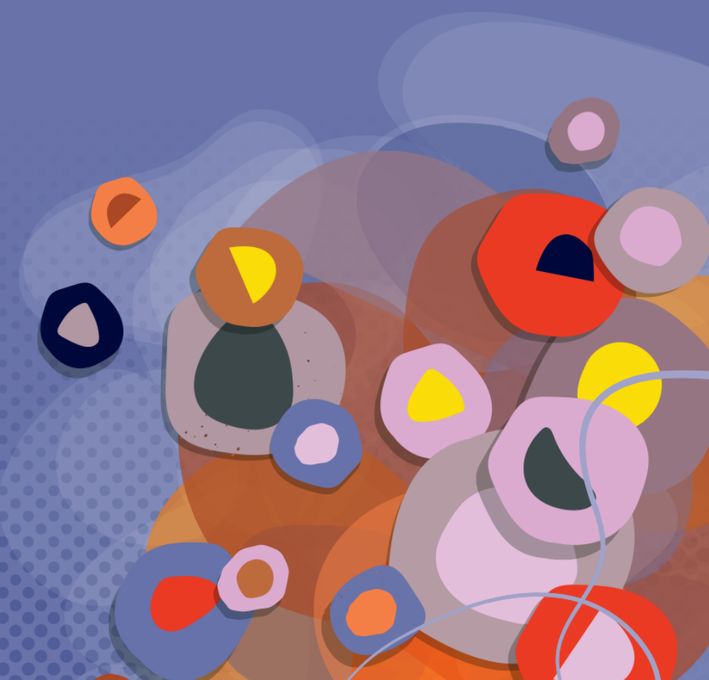
- Data will make or break your Machine Learning efforts and systems.
  - Think of it early, think of it often
  - Good data, is only as good as your org's ability to use it

- Cleaning your data has multiple facets:
  - Filtering, Removing Outliers and Normalizing Values
  - Centralizing your Database schemas can help with PII and the chaos of NoSQL databases.
  - Tackling these data architecture challenges becomes harder over time, bring someone early to the conversation.
- GOOD NEWS!
  - Your Business Analytics and Data teams will thank you for it

GDC
March 20-24, 2023
San Francisco, CA

Act 2

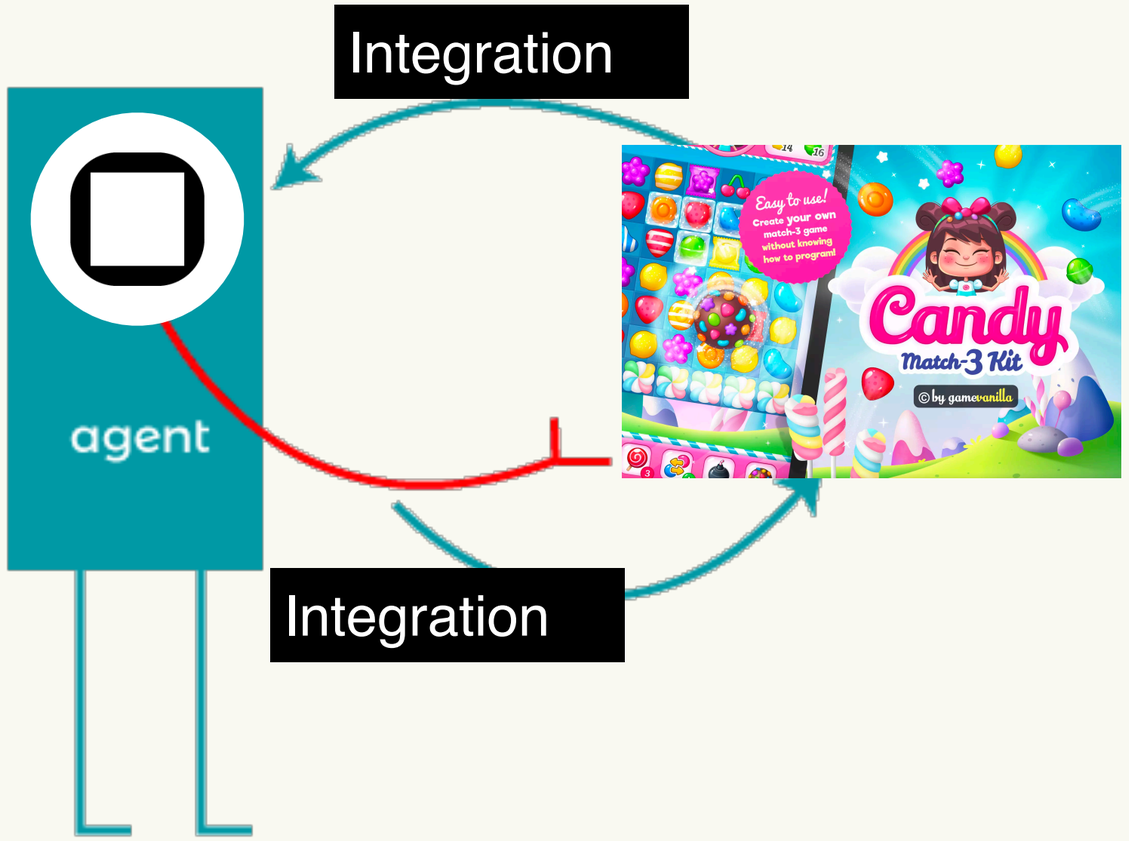
Sensing & Acting



Sensing & Acting in Candy Land

- Test bed to experiment and understand integration
- Closest we can get to an actual game without having you sign a NDA
- While we’re gonna look at some of it through a critical light - We have nothing for respect for gamevanilla

How This Works



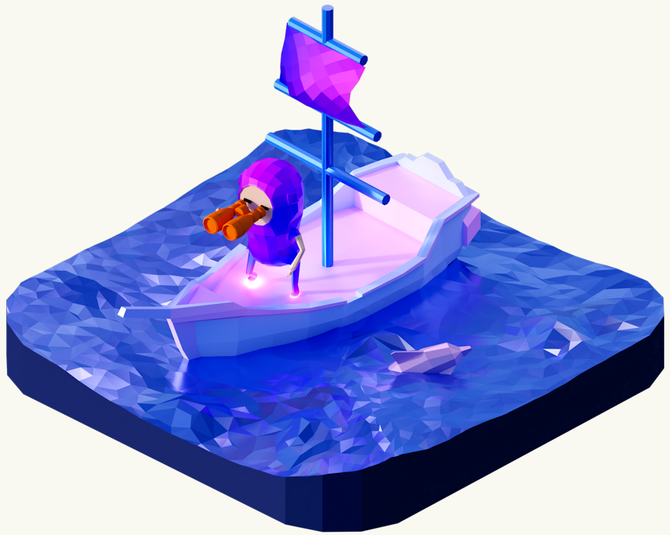
Integrating With Your Game

- The basics - Starting the game:
  - How many external dependencies does your game have? (Game Data Server, Login, CDNs, …)

- AI-Friendly Input Systems:
  - How easy is it for an automated system to manipulate your game state in a similar fashion that a player would?

- Observable States:
  - How easy is it to serialize your game state into a new format? (Yaml, JSON, RSON)

When to Act and Observe?

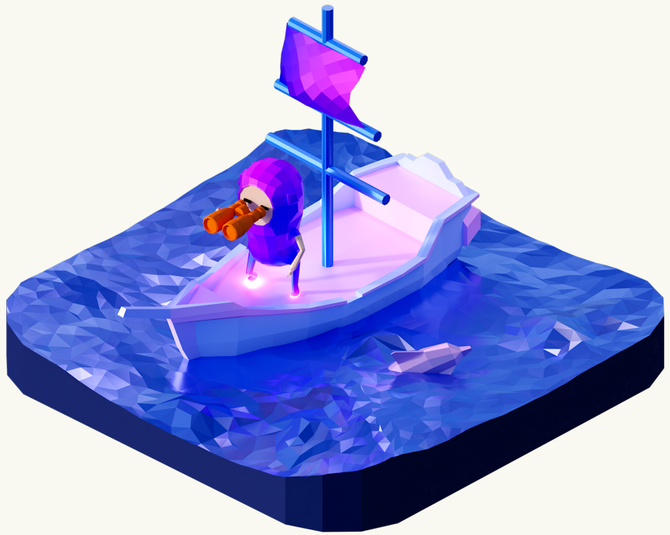


```
/// <summary>
/// Applies the gravity to the level tiles.
/// </summary>
public void ApplyGravity()
{
    StartCoroutine(ApplyGravityAsync());
}
```

```
            ExplodeTile(tile);
        }

        StartCoroutine(ApplyGravityAsync(0.5f));
    }
```

```
            }
            else if (consecutiveCascades == 6)
            {
                gameScene.ShowComplimentText(ComplimentType.Yummy);
            }
        }

        StartCoroutine(ApplyGravityAsync(didAnySpecialCandyExplode ? 0.5f : 0.0f));
        return true;
    }
    else
    {
        return false:
```

```
private IEnumerator ApplyGravityAsync(float delay = 0.0f)
{
    ClearSuggestedMatch();
    if (suggestedMatchCoroutine != null)
    {
        StopCoroutine(suggestedMatchCoroutine);
        suggestedMatchCoroutine = null;
    }

    inputLocked = true;
    yield return new WaitForSeconds(delay);
    ApplyGravityInternal();
    possibleSwaps = DetectPossibleSwaps();
    yield return new WaitForSeconds(1.0f);
```



…Actually do the move…

AI Friendly Input Systems

```
class MyGameObject : MonoBehaviour
{
    void Update()
    {
        if (UnityEngine.Input.GetKeyDown("space"))
        {
            // Space Stuff
        }
    }
}
```


```
class MyInputHandler : MonoBehaviour
{
    public bool spacePressed = false;
    void Update()
    {
        spacePressed = false;
        if (UnityEngine.Input.GetKeyDown("space"))
        {
            spacePressed = true;
        }
    }
}
```

```
class MyGameObject : MonoBehaviour
{
    public MyInputHandler InputHandler;
    void Update()
    {
        if (InputHandler.spacePressed)
        {
            // Space Stuff
        }
    }
}
```

Observing the Environment

- The GameObject and the Update loop approach to Game Engines make knowing when the state is 'ready to be observed', relatively challenging.

- Due to the need to make Engines more accessible or easier to iterate on, it is easy to blur concerns between game objects or their components - Which makes it hard to serialize or observe them.

Mixing Concerns - Basics

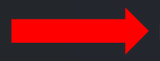
- Core Game shouldn’t depend on pre-Game State
  - Mockable Login
  - Mockable Inventory
  - Dismissible Takeovers

- Keeping these systems separated from your Core Game helps AI efforts
  - And also QA efforts

Mixing Concerns

```
if (!(tileA.GetComponent<Tile>().x != tileB.GetComponent<Tile>().x &&
      tileA.GetComponent<Tile>().y != tileB.GetComponent<Tile>().y))
{
    var selectedTileCopy = selectedTile;
    selectedTile.GetComponent<SpriteRenderer>().sortingOrder = 1;
    LeanTween.move(selectedTile.gameObject, hit.collider.gameObject.transform.position, 0.25f).setOnComplete(
        () =>
        {
            selectedTileCopy.GetComponent<SpriteRenderer>().sortingOrder = 0;
            gameScene.DisableBoosterMode();
            HandleMatches(true);
            ConsumeBooster(button);
        });
    LeanTween.move(hit.collider.gameObject, selectedTile.transform.position, 0.25f);

    var idxA = tiles.FindIndex(x => x == tileA);
    var idxB = tiles.FindIndex(x => x == tileB);
    tiles[idxA] = tileB.gameObject;
    tiles[idxB] = tileA;

    tileA.GetComponent<Tile>().x = idxB % level.width;
    tileA.GetComponent<Tile>().y = idxB / level.width;
    tileB.GetComponent<Tile>().x = idxA % level.width;
    tileB.GetComponent<Tile>().y = idxA / level.width;
```

Mixing Concerns



```
if (!(tileA.GetComponent<Tile>().x != tileB.GetComponent<Tile>().x &&
      tileA.GetComponent<Tile>().y != tileB.GetComponent<Tile>().y))
{
    var selectedTileCopy = selectedTile;
    selectedTile.GetComponent<SpriteRenderer>().sortingOrder = 1;
    LeanTween.move(selectedTile.gameObject, hit.collider.gameObject.transform.position, 0.25f).setOnComplete(
        () =>
        {
            selectedTileCopy.GetComponent<SpriteRenderer>().sortingOrder = 0;
            gameScene.DisableBoosterMode();
            HandleMatches(true);
            ConsumeBooster(button);
        });
    LeanTween.move(hit.collider.gameObject, selectedTile.transform.position, 0.25f);

    var idxA = tiles.FindIndex(x => x == tileA);
    var idxB = tiles.FindIndex(x => x == tileB);
    tiles[idxA] = tileB.gameObject;
    tiles[idxB] = tileA;

    tileA.GetComponent<Tile>().x = idxB % level.width;
    tileA.GetComponent<Tile>().y = idxB / level.width;
    tileB.GetComponent<Tile>().x = idxA % level.width;
    tileB.GetComponent<Tile>().y = idxA / level.width;
```

Mixing Concerns

```
if (!(tileA.GetComponent<Tile>().x != tileB.GetComponent<Tile>().x &&
      tileA.GetComponent<Tile>().y != tileB.GetComponent<Tile>().y))
{
    var selectedTileCopy = selectedTile;
    selectedTile.GetComponent<SpriteRenderer>().sortingOrder = 1;
    LeanTween.move(selectedTile.gameObject, hit.collider.gameObject.transform.position, 0.25f).setOnComplete(
        () =>
        {
            selectedTileCopy.GetComponent<SpriteRenderer>().sortingOrder = 0;
            gameScene.DisableBoosterMode();
            HandleMatches(true);
            ConsumeBooster(button);
        });
    LeanTween.move(hit.collider.gameObject, selectedTile.transform.position, 0.25f);

    var idxA = tiles.FindIndex(x => x == tileA);
    var idxB = tiles.FindIndex(x => x == tileB);
    tiles[idxA] = tileB.gameObject;
    tiles[idxB] = tileA;

    tileA.GetComponent<Tile>().x = idxB % level.width;
    tileA.GetComponent<Tile>().y = idxB / level.width;
    tileB.GetComponent<Tile>().x = idxA % level.width;
    tileB.GetComponent<Tile>().y = idxA / level.width;
```

Mixing Concerns

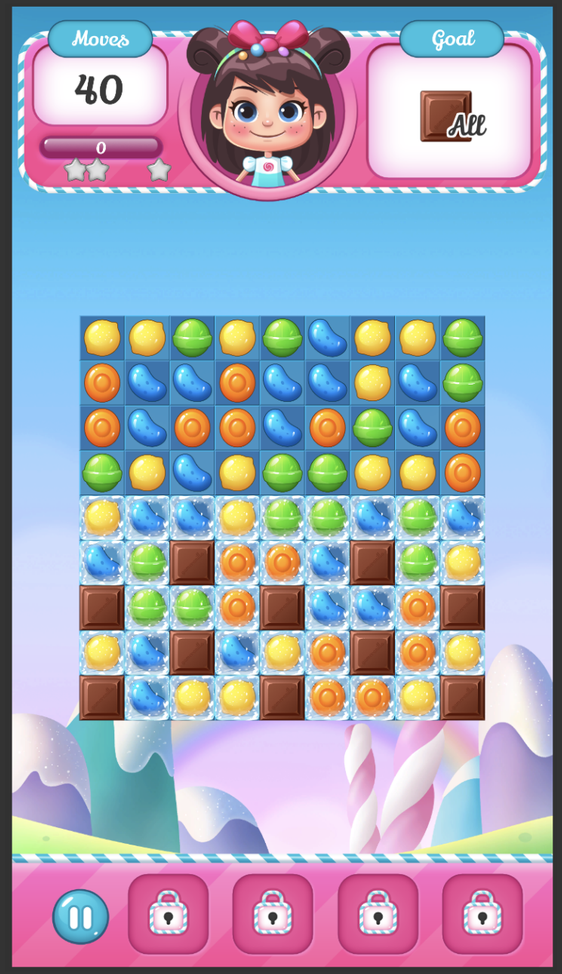
```
if (!(tileA.GetComponent<Tile>().x != tileB.GetComponent<Tile>().x &&
      tileA.GetComponent<Tile>().y != tileB.GetComponent<Tile>().y))
{
    var selectedTileCopy = selectedTile;
    selectedTile.GetComponent<SpriteRenderer>().sortingOrder = 1;
    LeanTween.move(selectedTile.gameObject, hit.collider.gameObject.transform.position, 0.25f).setOnComplete(
        () =>
        {
            selectedTileCopy.GetComponent<SpriteRenderer>().sortingOrder = 0;
            gameScene.DisableBoosterMode();
            HandleMatches(true);
            ConsumeBooster(button);
        });
    LeanTween.move(hit.collider.gameObject, selectedTile.transform.position, 0.25f);

    var idxA = tiles.FindIndex(x => x == tileA);
    var idxB = tiles.FindIndex(x => x == tileB);
    tiles[idxA] = tileB.gameObject;
    tiles[idxB] = tileA;

    tileA.GetComponent<Tile>().x = idxB % level.width;
    tileA.GetComponent<Tile>().y = idxB / level.width;
    tileB.GetComponent<Tile>().x = idxA % level.width;
    tileB.GetComponent<Tile>().y = idxA / level.width;
```

Mixing Concerns - Some Solutions

- Subordinate your Animation to your Logic - Not the other way around.
  - MVC can help here
  - “Dumb” Presentation Layer - AI will not look at the UI

- Data Driven Architecture is very AI friendly
  - ECS can help here (like DOTS from Unity)

Lost in Translation...




```
%TAG !u! tag:unity3d.com,2011:
--- !u!1 &1193228960034462
GameObject:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  serializedVersion: 6
  m_Component:
  - component: {fileID: 4601373430386348}
  - component: {fileID: 114523015657736830}
  - component: {fileID: 114440977651962760}
  - component: {fileID: 114903634578139318}
  - component: {fileID: 8393762374588120423}
  m_Layer: 0
  m_Name: GameManager
  m_TagString: Untagged
  m_Icon: {fileID: 0}
  m_NavMeshLayer: 0
  m_StaticEditorFlags: 0
  m_IsActive: 1
--- !u!4 &4601373430386348
Transform:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 1193228960034462}
  m_LocalRotation: {x: 0, y: 0, z: 0, w: 1}
  m_LocalPosition: {x: 0, y: 0, z: 0}
  m_LocalScale: {x: 1, y: 1, z: 1}
  m_Children: []
  m_Father: {fileID: 0}
  m_RootOrder: 0
  m_LocalEulerAnglesHint: {x: 0, y: 0, z: 0}
--- !u!114 &114523015657736830
MonoBehaviour:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 1193228960034462}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: 41e0d37bf3089ad4d84c34d3fb1b7708, type: 3}
  m_Name:
  m_EditorClassIdentifier:
  gameConfig:
    maxLives: 0
    timeToNextLife: 0
    livesRefillCost: 0
    initialCoins: 0
    numExtraMoves: 0
```

```
[ [ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ],
  [ 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1 ] ],
```

Observable GameObjects

```
class MyGameObject : MonoBehaviour
{
    public Vector3 Field1;
    public string  Field2;
    public int     Field3;
}
```

```
class MyGameObject : MonoBehaviour
{
    [Serializable]
    public struct DataModel
    {
        public string Field2;
        public int    Field3;
    }

    public Vector3 Field1;
    public DataModel Data;
}
```

A Note About Observing Pixels

- While there have been successes, they wont solve every problem and are much more expensive to deploy.

- We will shift part of the issues to the Render Pipeline.
  - Can you turn off some of the effects?
  - Can you remove text or the entire HUD?

A Note on Determinism

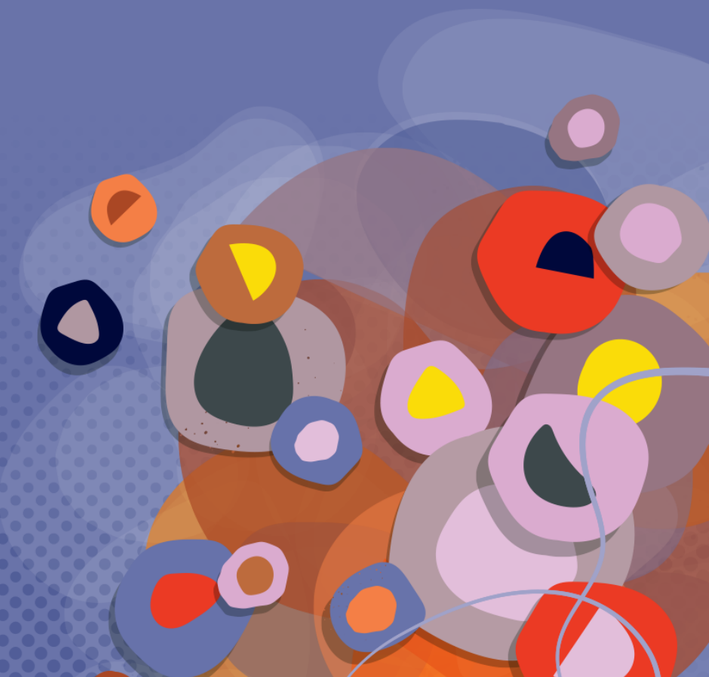
- The Suggestion Issue in Candy Land
  - Showcases what happens when multiple systems ‘compete’ for RNG
- How do we fix it?
  - “Scoped” Random Number Generators and Seeding
  - Akin to Unreal’s RandomStreams

Summary of Act II

- Integration Challenges:
  - Input & Output abstraction layer
  - Serialization is a huge pain

- Lessons from the field:
  - QA Tooling Investment paid off
  - Good vs Perfect

Epilogue - A Bright Future



Paradigm Shift

Let there be games…

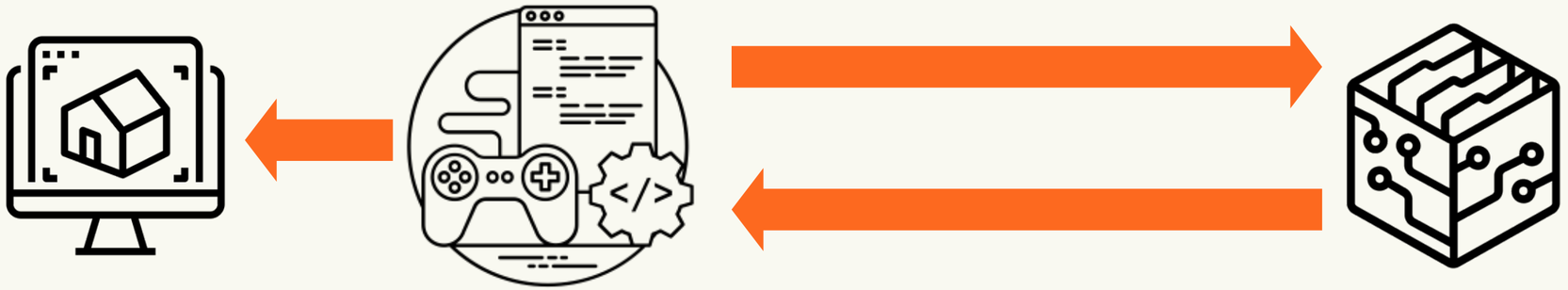
We did everything together:

On the second generation…

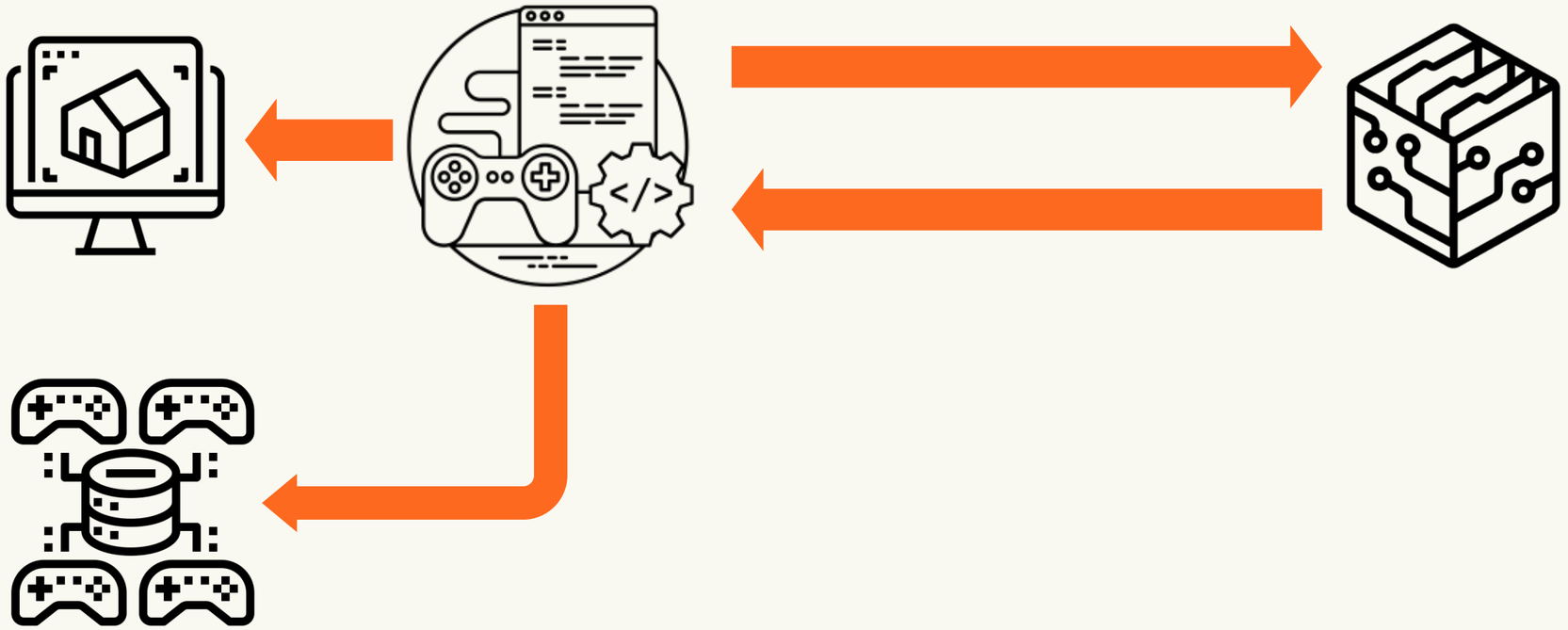
We created a render loop

Then we had game engines

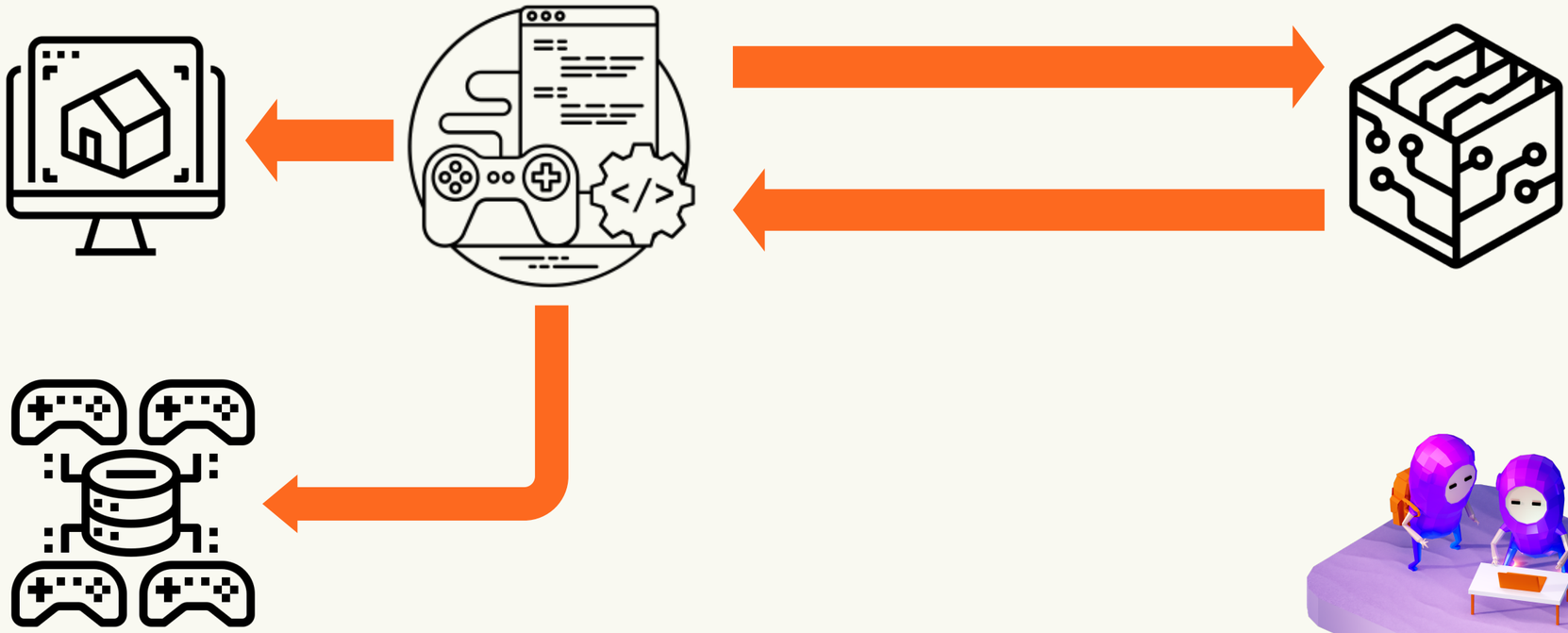
Things we repeat



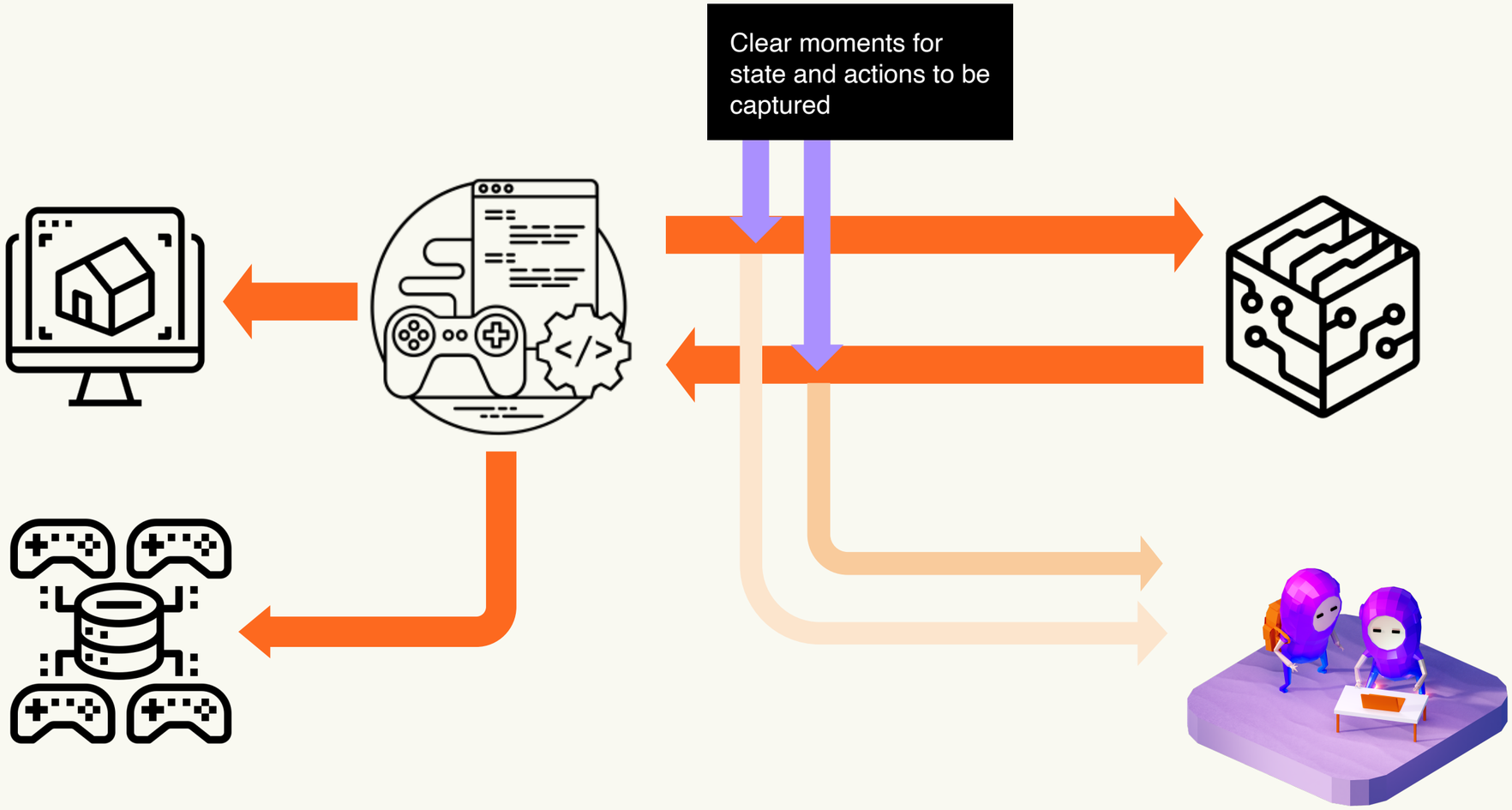
Modern Multiplayer Game Architecture



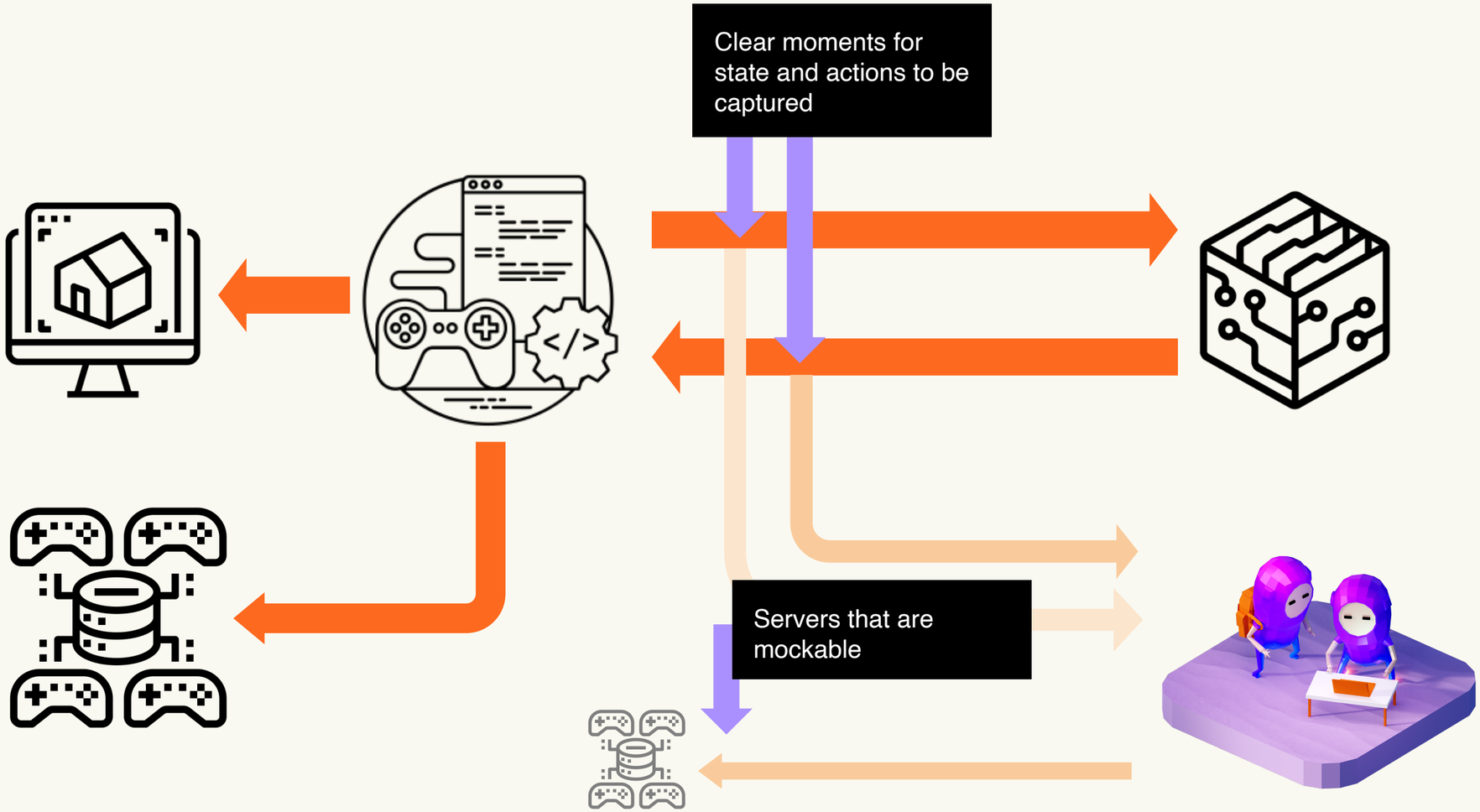
What Would Make our Bean Happy?



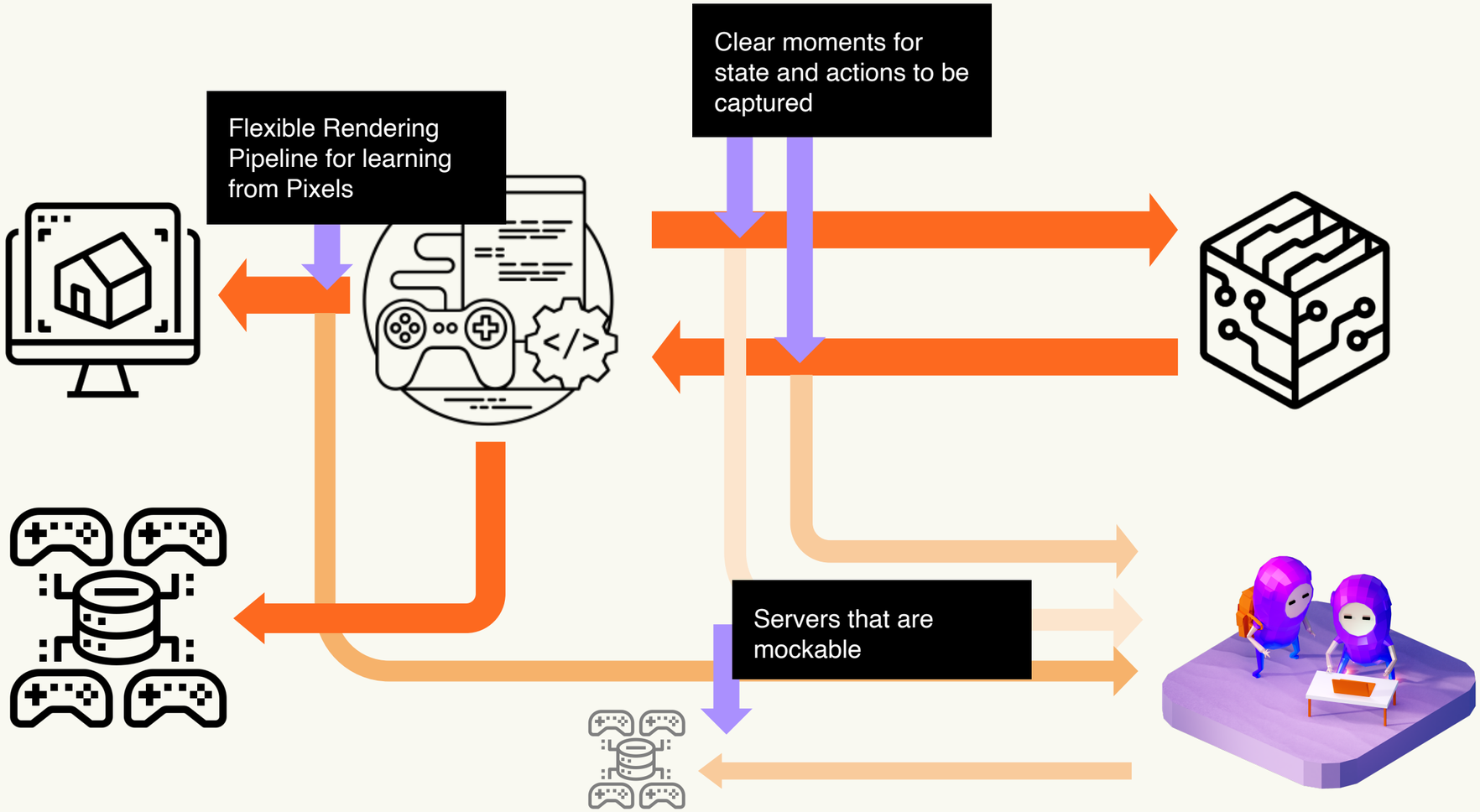
What Would Make our Bean Happy?



What Would Make our Bean Happy?



What Would Make our Bean Happy?



Recap

- Accessible, clean and well architected data is crucial to the success of your ML Systems.
- AI Friendly Inputs and Observable Game Worlds require small, but deliberate, adjustments in how we design our code.
- Pixels are an interesting avenue, but will come with their own set of challenges.
- Most of the suggestions here can be applied incrementally, and even if you end up doing no Machine Learning they will still yield value in other places.

Questions? Comments? Curses?

Ricardo Sisnett
sisnett@modl.ai

sysnet_ai

sysnet-ai