GTC

Towards Advanced Automated Game Testing with AI

Linus Gisslen – SEED
Konrad Tollmar – SEED

Presenters

Linus Gisslén
Sr Research Engineer
SEED / EA

Konrad Tollmar
Research Director SEED / EA
Associate Professor KTH

Outline

The Talk & Who We Are?

Introduction to EA and SEED

What is video games testing?

Visual testing: Graphical glitches

Game play testing with RL: Possibilities and Challenges

Adversarial RL for Procedural Content Generation

Using curiosity for better game coverage

Future Work & Challenges

Summary

Q&A

SEED // SEARCH FOR EXTRAORDINARY EXPERIENCES DIVISION

SEED is a cross-disciplinary R&D team
We explore the future of interactive entertainment

Future Graphics

Cinema-quality
& artist-friendly
content at Scale

Hybrid Ray Tracing

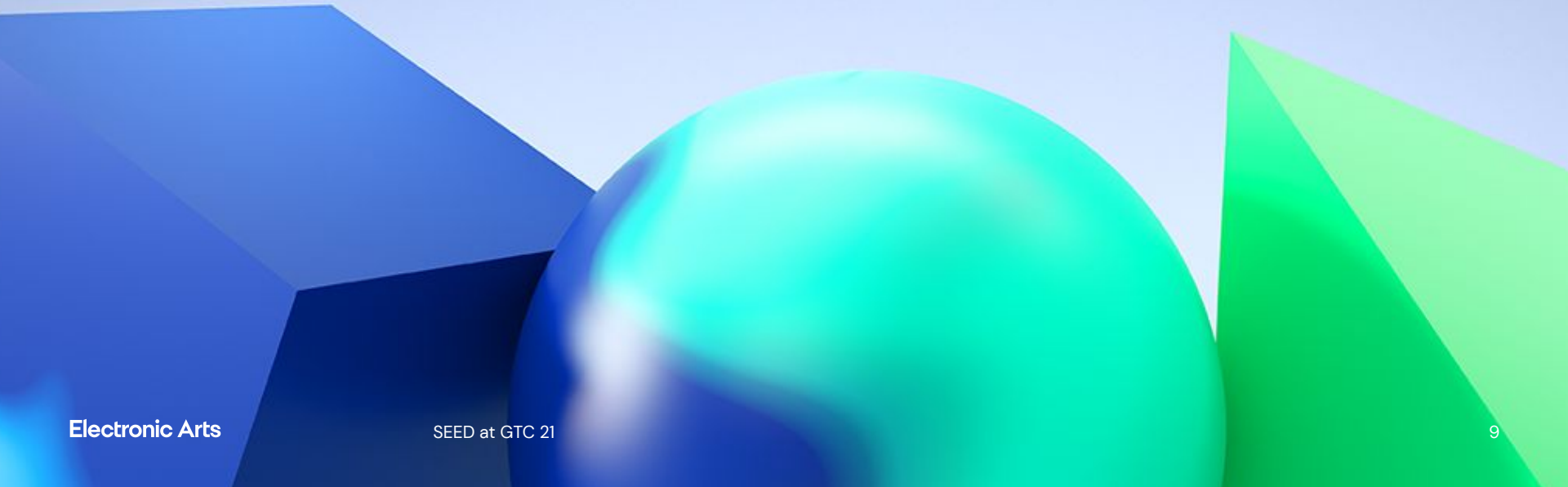
Advanced Avatars

End-to-end pipeline for visually convincing avatars

Deep Testing

Human-like behavior
for improving
game testing

Deep Testing



Why do we test games?

Examples of bugs/exploits:

Player stuck

Missing collision box -> Player walks through walls

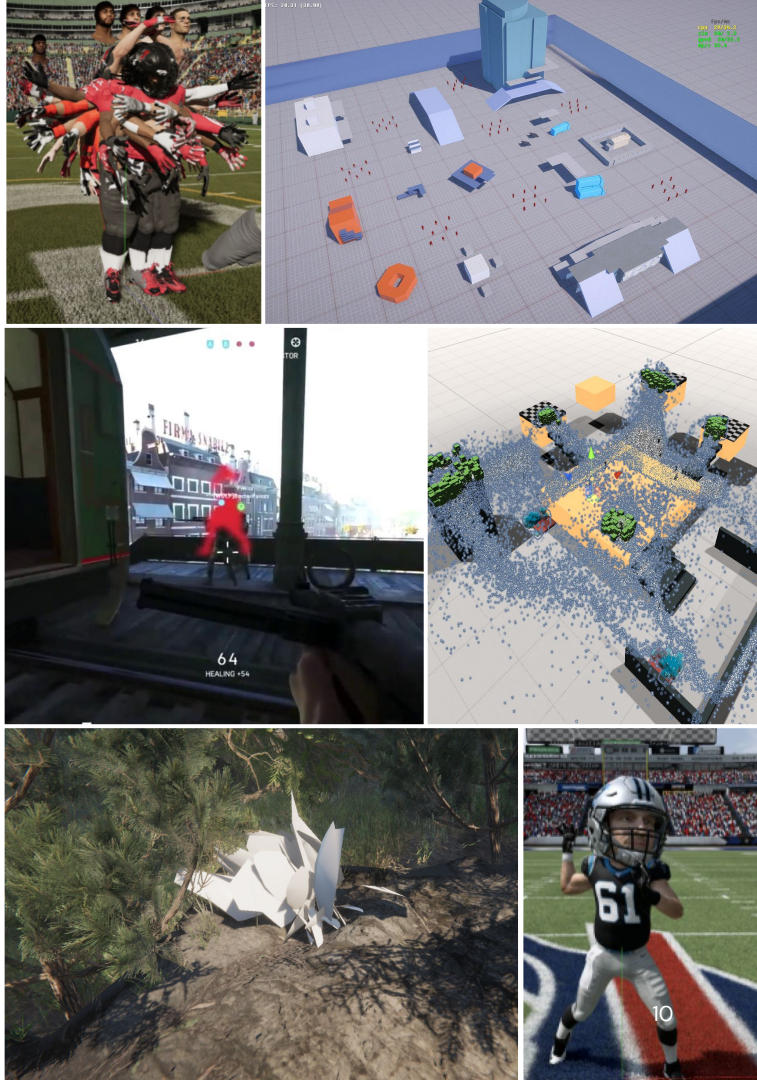
Players falling through the ground

Imbalances (maps, characters, etc.)

Visual bugs

Load test (map coverage)

Etc.



Motivation: Automated vs. human playtesting

Automated testing

Faster testing -> Shorter time to market

Less shipped bugs

Less “exponential growth” cost in testing

with growing games/live services

Less manual/mundane labor

More time for “meaningful” testing

Happier employees

Reduces risk

Goal with our research

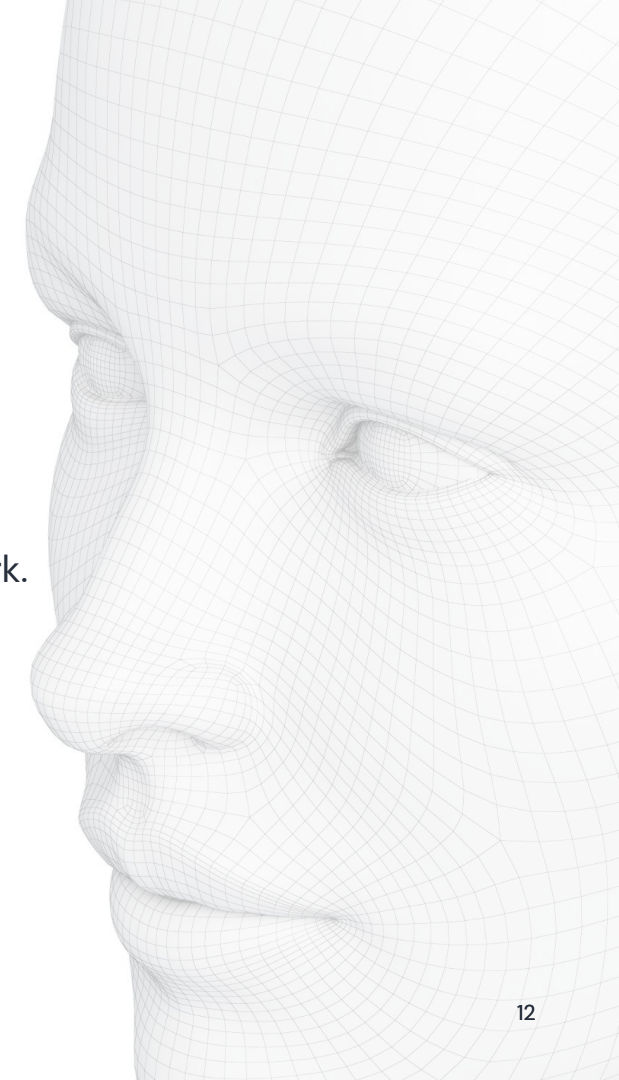
In short:

Research human-like automated game testing
as we want to test as closely to how players perceive our games.

Use ML to accelerate game testing at scale & breadth, freeing human playtesters from repetitive tasks, moving towards more meaningful work.

Research AI-powered methods to help developers find bugs faster, and improve overall product quality, in real-time.

Use testing as a springboard to develop ML-based AI in games.

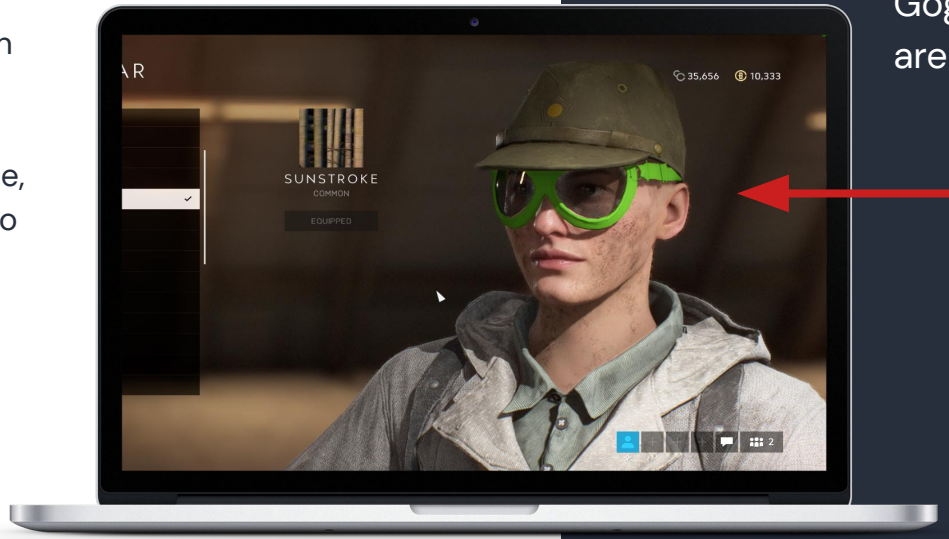


Visual Testing & Validation

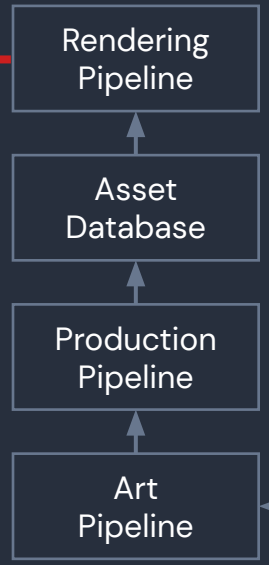
What leads to a graphical issue/bug?

Any error in the steps within the rendering pipeline, production pipeline, art pipeline, and asset database, etc. could potentially lead to rendered glitches.

Basically it's an aggregation of everything that can possibly go wrong

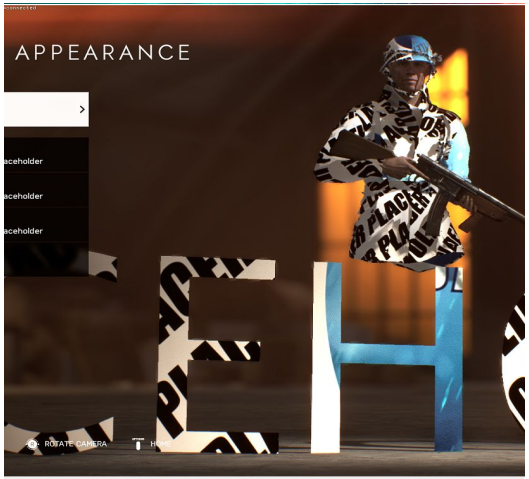


Result:
Goggles
are missing textures



Example From Production

Jira ticket search: 20k+ on graphical issues for one game



Replacement/Missing texture

When an texture is missing the default texture is the fallback

Stretched texture

Stretching of textures are commonplace

Low res texture

Compared to surrounding texture the texture has a lower resolution

2 Ways For Getting Data For Training

Gather in-game graphical glitches

Pros
Closer to “real” data

Cons
Fewer samples

No/little control over data

Hard/slow to create new data sets

Risk of biased data set

Unbalanced

Generating synthetic data with known glitches

Pros
Control over data: objects, lightning, rotation, background

Automate data gathering: can easily gather 100k+ images in a short time

Can be generated with game assets in game engine

Balanced

Cons
Not “real” data

Training Samples

Electronic Arts

Model

Selecting Approaches

1. Unsupervised Learning

Good for seeing previously un-seen errors

Ideal, but very hard to train and not practical yet

2. Anomaly detection approaches

Good for finding “outliers” i.e. something completely unexpected

Not practical

3. Supervised Learning: Image detection <- Our pick

Control over data makes this approach very effective

Types:

- Object detection - Not beneficial
- Semantic segmentation - Very computationally costly
- Classification <- Our pick

To summarize: Keep it simple!

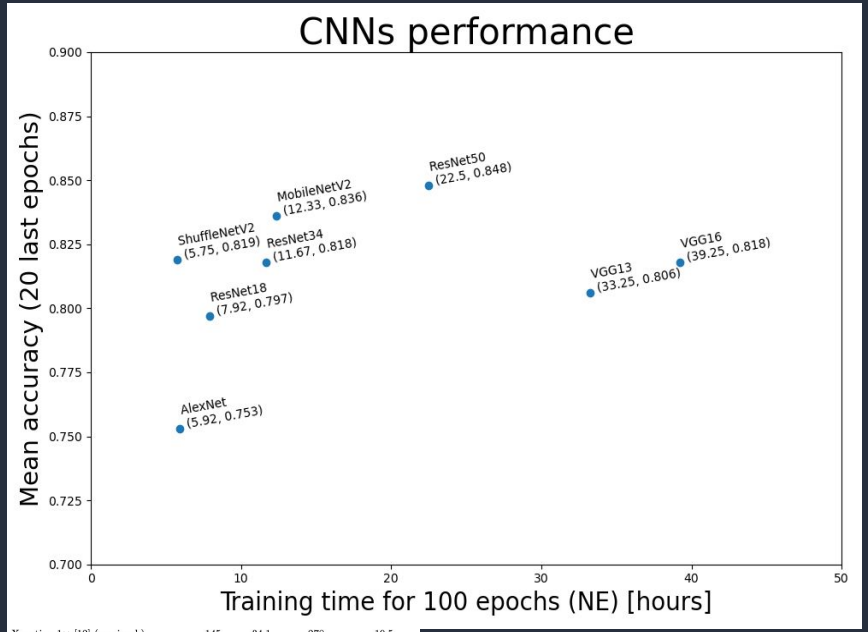
Model

There are 200+ SL architectures, we studied 5 architectures:

1. AlexNet (2012) Input 256x256, 60M parameters
2. VGG-16 (2014) Input 224x224, 133M parameters
3. ResNet 18-50 (2015) Input 224x224, 11M-25.6M parameters
4. ShuffleNetV2 (2016) Input 224x224, 7.4M parameters
   Our pick
5. MobileNet (2019) Input 224x224, 4.5M parameters

Training:

- 1 Machine with dual GeForce GTX 1080 Ti.
- <10h training time (Shufflenet)
- Inference/run-time on ShuffleNet 60fps.



Xception 1× [12] (our impl.)	145	34.1	278	19.5
IGCV2-0.5 [27]	156	34.5	132	15.5
IGCV3-D (0.7) [28]	210	31.5	143	11.7
ShuffleNet v2 1.5× (ours)	299	**27.4**	255	**11.8**
0.75 MobileNet v1 [13]	325	31.6	**314**	10.6
1.0 MobileNet v2 [14]	300	28.0	180	8.9
1.0 MobileNet v2 [14] (our impl.)	301	28.3	180	8.9
ShuffleNet v1 1.5× (g=3) [15]	292	28.5	164	10.3
DenseNet 1.5× [6] (our impl.)	295	39.9	274	9.7
CondenseNet (G=C=8) [16]	274	29.0	-	-
Xception 1.5× [12] (our impl.)	305	29.4	219	10.5
IGCV3-D [28]	318	27.8	102	6.3
ShuffleNet v2 2× (ours)	591	**25.1**	217	**6.7**
1.0 MobileNet v1 [13]	569	29.4	**247**	6.5
1.4 MobileNet v2 [14]	585	25.3	137	5.4
1.4 MobileNet v2 [14] (our impl.)	587	26.7	137	5.4
ShuffleNet v1 2× (g=3) [15]	524	26.3	133	6.4
DenseNet 2× [6] (our impl.)	519	34.6	197	6.1
CondenseNet (G=C=4) [16]	529	26.2	-	-
Xception 2× [12] (our impl.)	525	27.6	174	**6.7**
IGCV2-1.0 [27]	564	29.3	81	4.9
IGCV3-D (1.4) [28]	610	25.5	82	4.5
ShuffleNet v2 2x (ours, with *SE* [8])	597	**24.6**	161	**5.6**
NASNet-A [9] (4 @ 1056, our impl.)	564	26.0	130	4.6
PNASNet-5 [10] (our impl.)	588	25.8	115	4.1

For more details see:

"Using Deep Convolutional Neural Networks to Detect Rendered Glitches in Video Games"
Garcia Ling et. al AIIDE-2020

Results

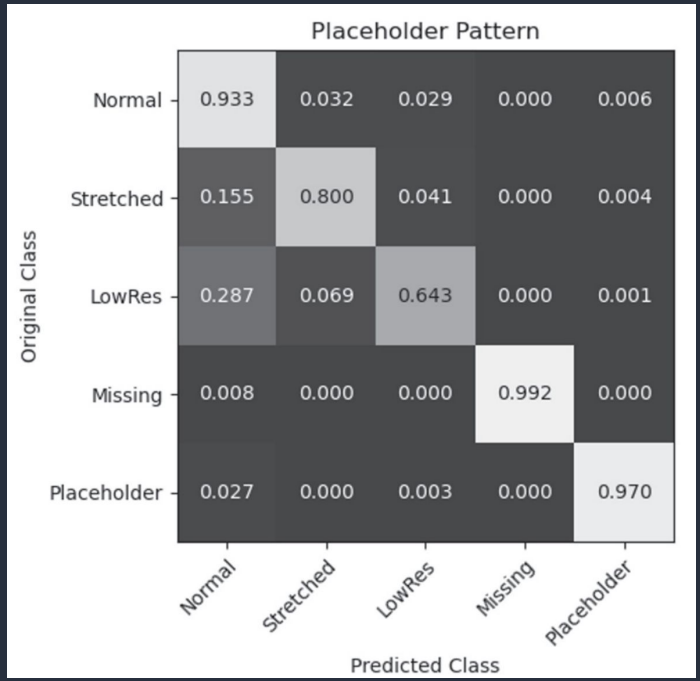
Previously seen objects, but in different environment, shadowing, lightning, angles, etc.

General: Binary accuracy of 86.8%, detecting 88% of the glitches with a false positive rate of 8.7%

Multi-class classification breakdown (see figure)

Good performance with missing/placeholder textures

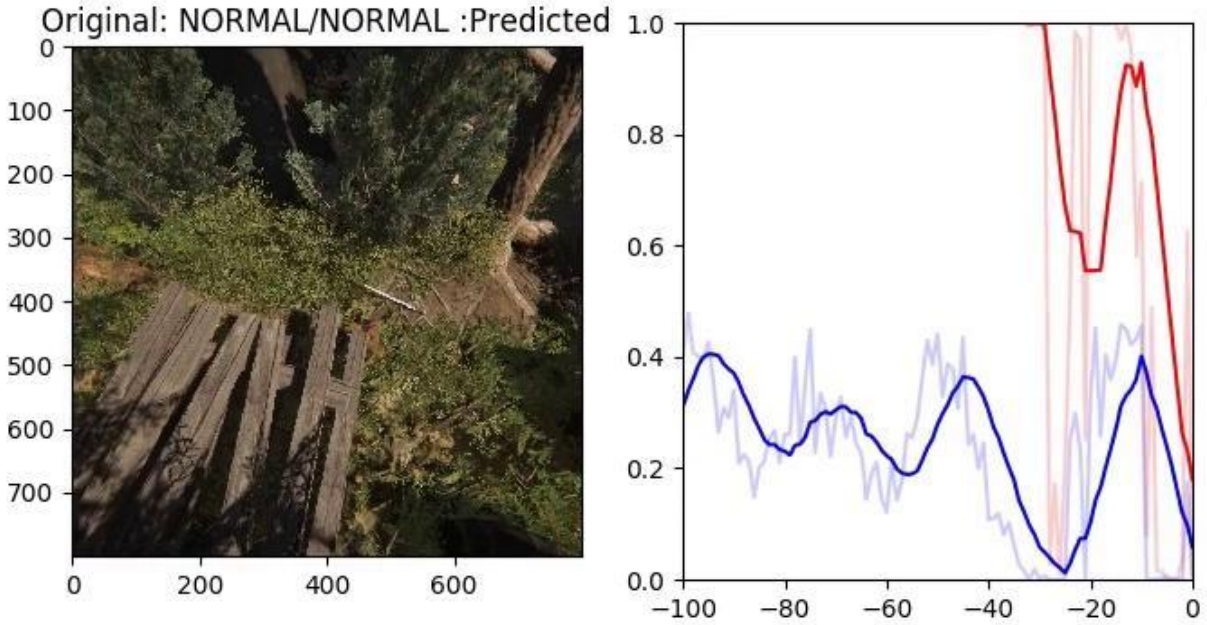
Some confusion with Corrupted (Stretched + LowRes) textures



For more details see:

"Using Deep Convolutional Neural Networks to Detect Rendered Glitches in Video Games"
Garcia Ling et. al AIIDE-2020

Results


Original: NORMAL/NORMAL :Predicted
0
100
200
300
400
500
600
700
0
200
400
600
1.0
0.8
0.6
0.4
0.2
0.0
−100
−80
−60
−40
−20
0


Gameplay testing with
Reinforcement Learning

Motivation for RL in Game Play Testing

"In **Battlefield V** testing all maps and modes for 1 hour requires 2304h of testing. 288 people to test that every day. If we add more maps and modes this number will be larger." Jonas Gillberg DICE (An EA Studio)

RL learns from interactions with the environment therefore:

It can find exploits without explicitly being "told".

Training a policy using reward functions can be more intuitive than scripting.
Also:

- Can learn things that are not scriptable.
- Allows to test games without extensive scripting.
- No need to rewrite scripts every time the environment changes

Learning to play a game requires exploration, adapting, trying stuff, etc. which gives a more "human like" testing and better coverage.

Scale and speed - Play hundreds of games simultaneously.

An RL trained agent that learned to exploit the game

Challenges: Our Research Directions

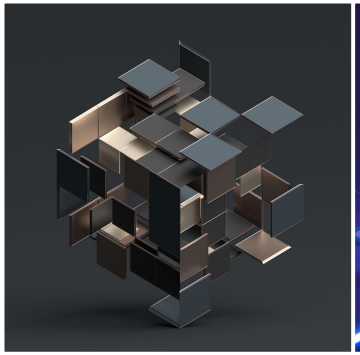
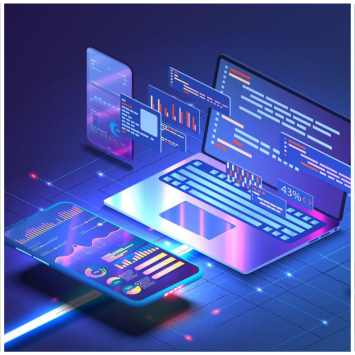
Generalization

Interpretation: Analytics

Control & Human-like Behavior

Fine-tuning of Behavior

Skill Level


Problem Statement Generalization

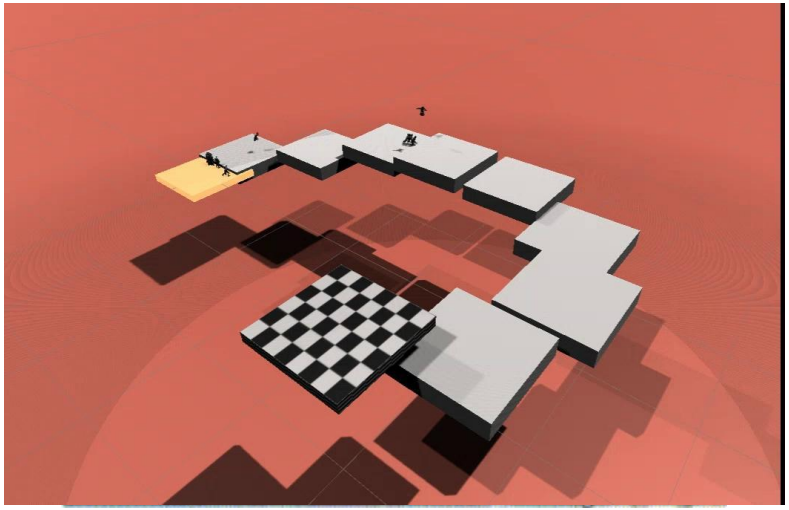
Training RL agents on a fixed map -> Memorization ->
No Generalization -> We need to re-train every new environment

Good for some problems but not all.
RL is notoriously hard to train for generalization.

Use-cases for improved generalization:

- In testing/playing previously unseen maps.
- Real-time applications where there is not time to retrain.
- Guided Content generation. It's better to find bugs directly in production rather than later

Good generalization is necessary!



Example on bad generalization ability

Problem Statement Generalization

RL agent does not generalize well when trained on static levels

One solution is to procedurally generate new levels to train to increase generality (2)

PCG only is no guarantee it is a good training environment with challenging yet not impossible maps

Proposed Solution:

Let another RL create the map and with observation and feedback from the solving RL it can learn to make difficult but not impossible maps

2) Increasing Generality in ML through PCG. *Risi et. al CoG 2020*

The Model

Adversarial RL for Procedural Content Generation (ARLPCG)

Requirements for training a good RL Agent:

Environments that are neither impossible nor trivial

Diverse environment so that the RL Agents becomes more robust to novel situations

Control over generation (bonus)

1. Generator + Solver Approach

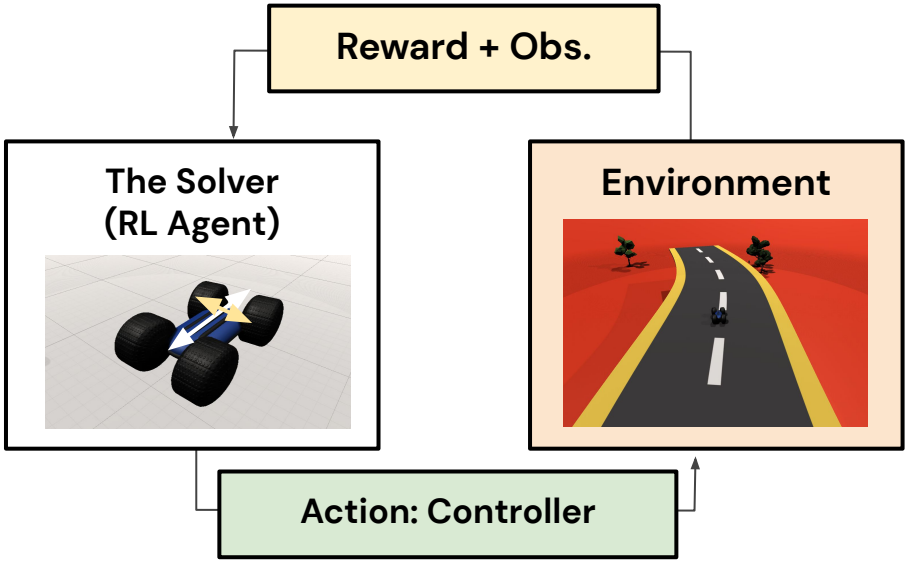
Ensures that environments are not impossible but at the same time challenging

2. Auxiliary input

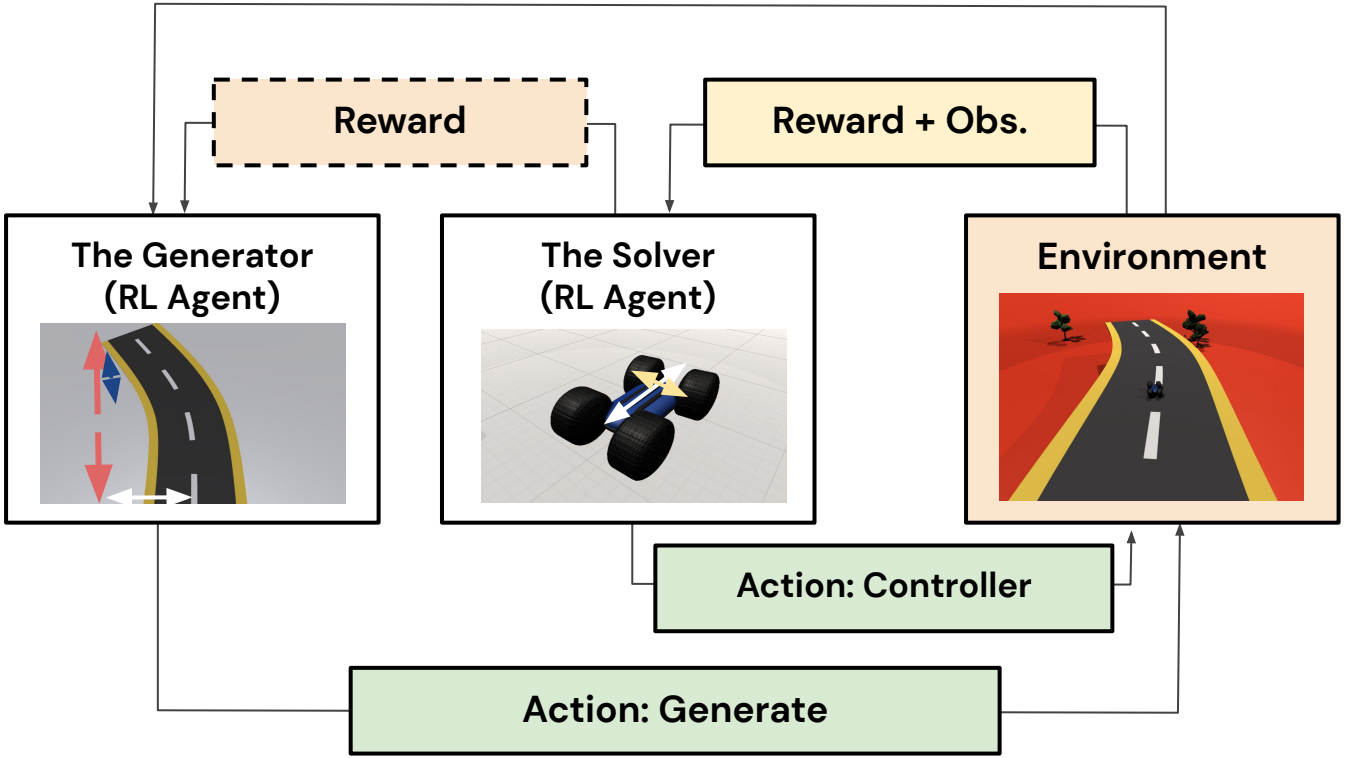
Introduces control over generation

Varying auxiliary input -> Diverse environment

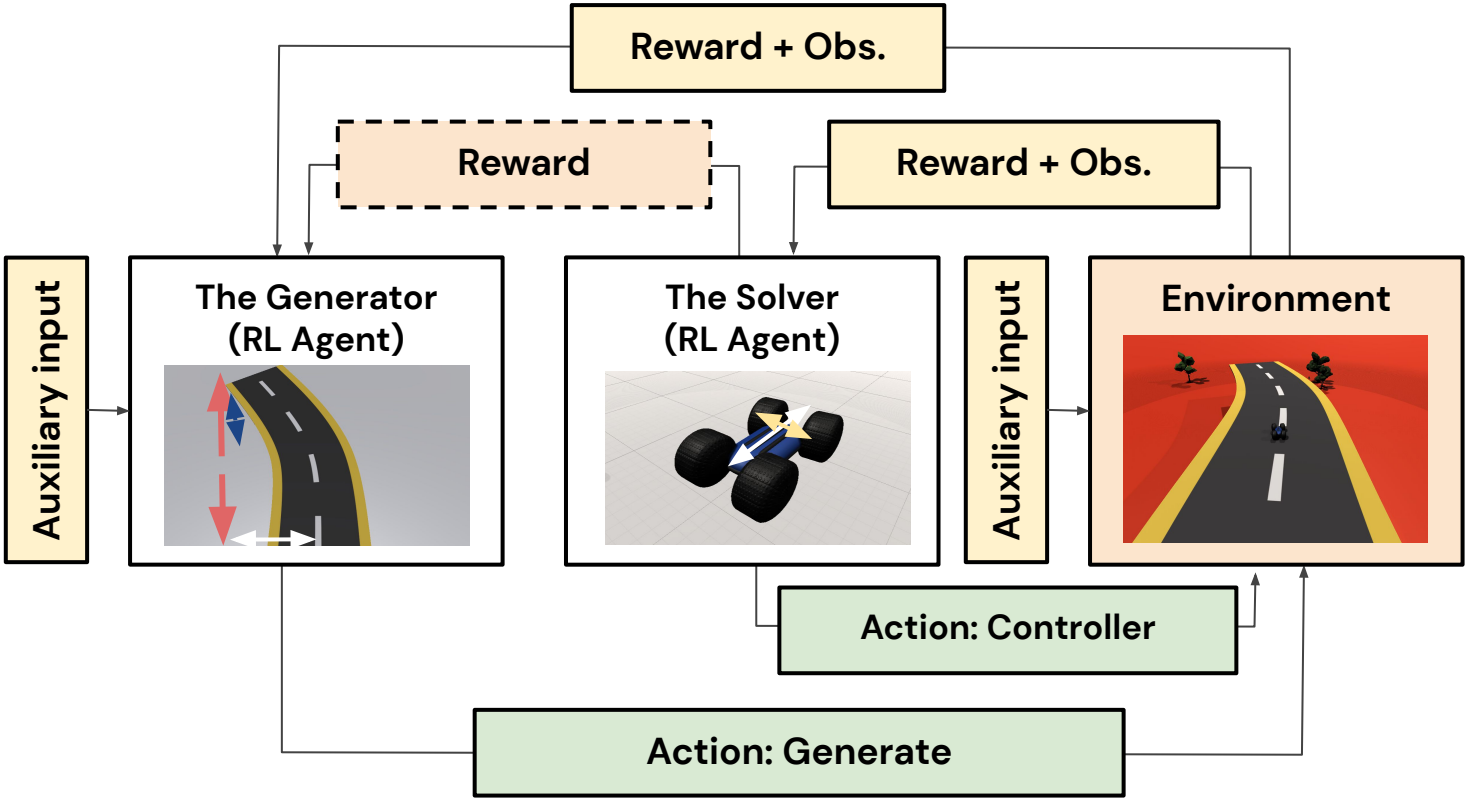
Architecture: Reinforcement Learning



Architecture: RL Solver & RL Generator

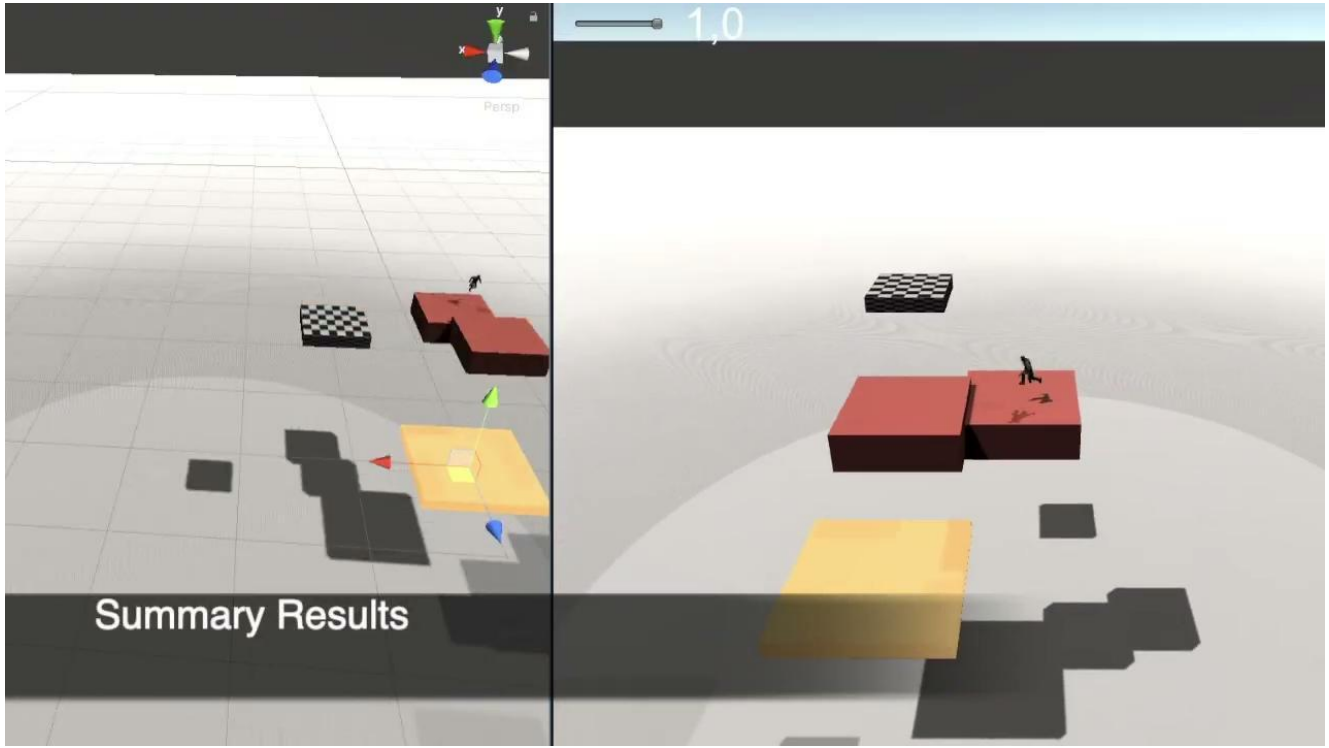


Architecture: ARLPCG



Generalization: Adversarial RL for PCG

Results published in Conference on Games (CoG) 2021



For more details see:

"Adversarial Reinforcement Learning for Procedural Content Generation"
Gisslén et. al CoG-2020

Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents

Camilo Gordillo
Joakim Bergdahl
Konrad Tollmar
Linus Gisslén

Exploration

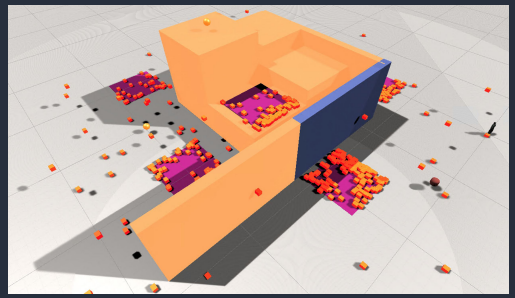
Problem Statement

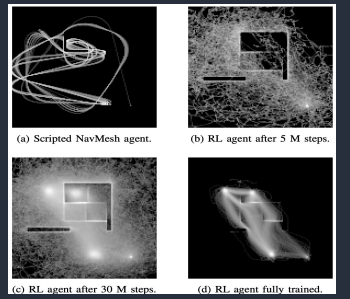
Good exploration leads to better game state coverage

Good exploration leads to less biased game state visits

Without exploration we will miss a lot of bugs that are on the edges of the environment

Good exploration is necessary! (1)





Pictures taken from (1)

1) **Augmenting Automated Game Testing with Deep Reinforcement Learning** Bergdahl et. al CoG 2020

Summary:

- With RL and scripted often exploration is driven by randomness
- RL improves game state coverage compared to scripted bots, but..
- Introducing curiosity lead to even more purposeful exploration and better test coverage (2)

2) **Improving Playtesting Coverage via Curiosity Driven RL agents** Gordillo et. al CoG 2021

Paper

Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents

Novelty seeking RL (Curiosity)

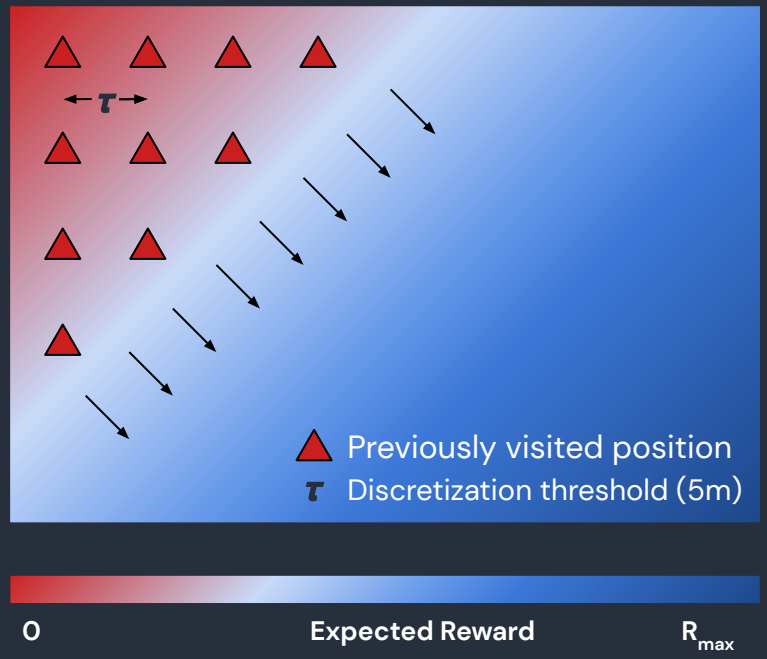
Encouraging RL agents to maximize coverage:

- Classical Proximal Policy Optimization (PPO) algorithm
- Reward proportional to the transition's novelty
  - Low and decreasing reward for visiting previously explored locations
  - High reward for reaching new areas or triggering new events

Data collection <u>while training takes place</u>

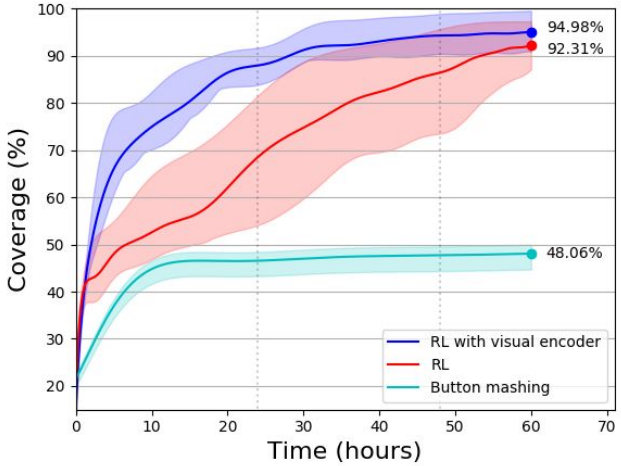
- The final model is not important (and maybe even useless)
- The collected data, however, can be very useful!

Playtesting through data analysis



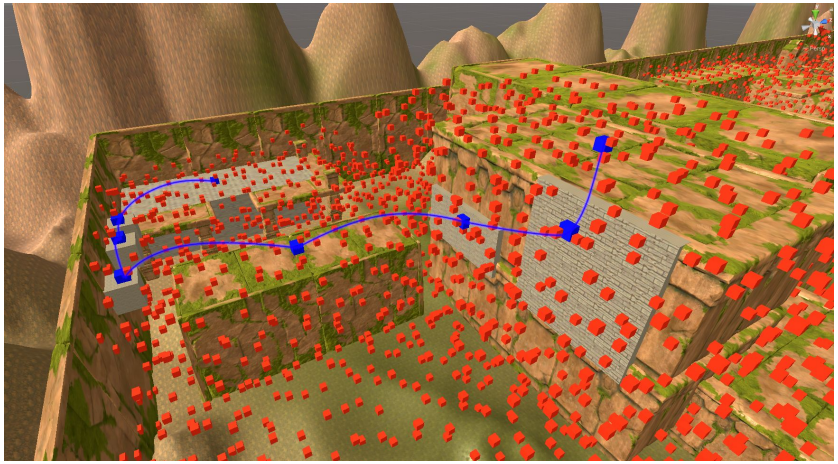
Results Coverage

Deploying **320 agents**
~90% coverage in about 24 hours

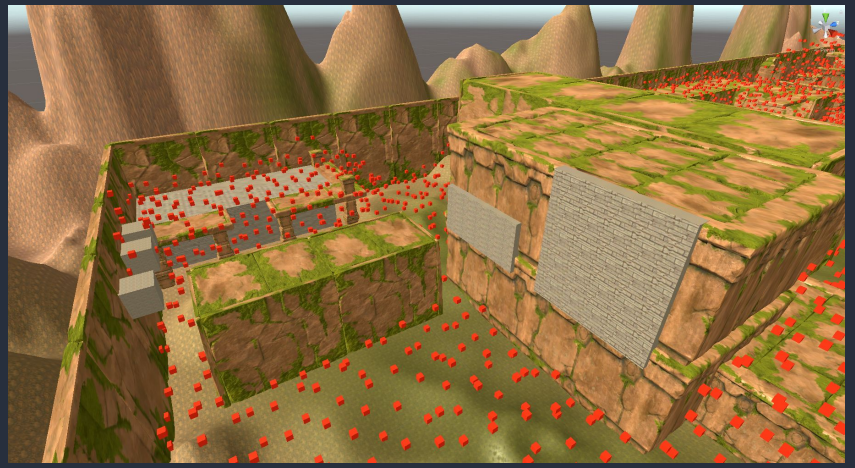


Results Coverage

RL policy



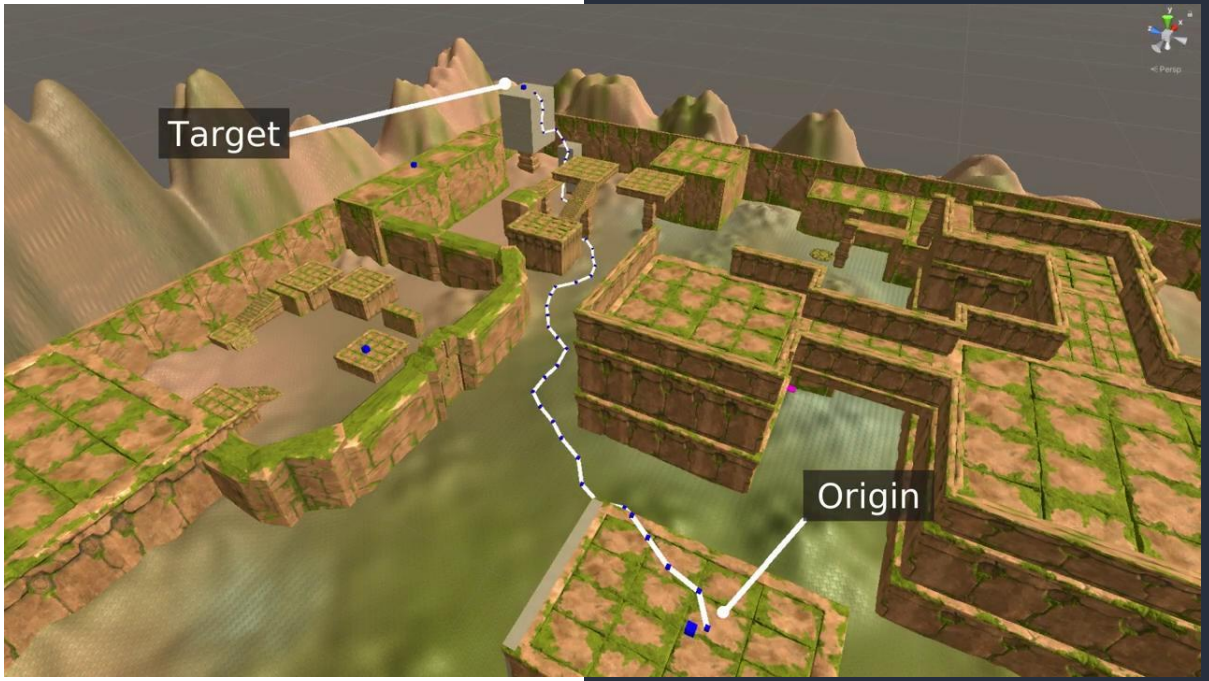
Random policy



Simple navigation strategies are unlikely to solve complex navigation challenges

Results

Connectivity Graph




Summary

Summary

Machine Learning has the potential to greatly improve automated testing as well as game-AI

Simple techniques like CNN for glitch detections works well and could be applied today

RL agents can find exploits without explicitly being "told" (ie. scripted)

Learning to play a game requires exploration, adapting, trying stuff, etc. which gives a more **"human like" testing and better coverage.**

RL agents can not only be used to play and test games **but also to generate them**

Data analysis and proper visualizations are key to make sense of the outcome of these experiments

Future Work & Challenges

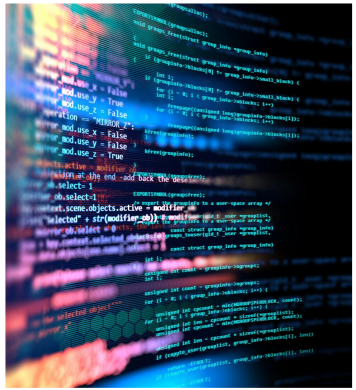
Going from Research to Development: Integration into existing game development pipeline

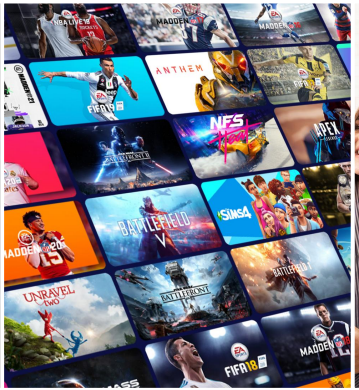
Scaling on breadth (across different games) and height (local -> cluster)

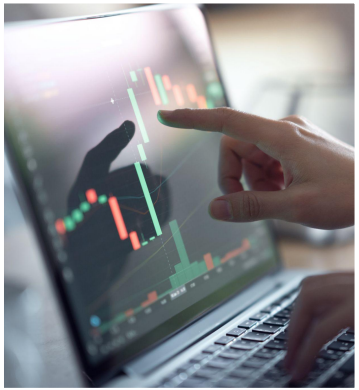
Different Personas for better understanding how different players play the same game

Generalization and fine tuning of skills so we do not have to re-train

Analytics and automated reporting of found bugs







Thank You / Q&A

...and yes we are hiring!
https://www.ea.com/seed

Contact
ktollmar@ea.com
lgisslen@ea.com