Horizontal Scaling with a Framework for Providing AI Solutions within a Game Company

John F. Kolen, Mohsen Sardari, Marwan Mattar, Nick Peterson, Meng Wu
EADP Intelligent Systems
Electronic Arts
209 Redwood Shores Parkway
Redwood City, California 94069
{jkolen,msardari,mmattar,npeterson,mewu}@ea.com

Abstract

Games have been a major focus of AI since the field formed seventy years ago. Recently, video games have replaced chess and go as the current "Mt. Everest Problem". This paper looks beyond the video games themselves to the application of AI techniques within the ecosystems that produce them. Electronic Arts (EA) must deal with AI at scale across many game studios as it develops many AAA games each year, and not a single, AI-based, flagship application. EA has adopted a horizontal scaling strategy in response to this challenge and built a platform for delivering AI artifacts anywhere within EA's software universe. By combining a data warehouse for player history, an Agent Store for capturing processes acquired through machine learning, and a recommendation engine as an action layer, EA has been delivering a wide range of AI solutions throughout the company during the last two years. These solutions, such as dynamic difficulty adjustment, in-game content and activity recommendations, matchmaking, and game balancing, have had major impact on engagement, revenue, and development resources within EA.

Introduction

Game playing has a long history within the AI community. Chess and checkers were the focus of early foundational work (e.g. (Turing 1953) and (Samuel 1959)) and have become, along with *Jeopardy!* (Ferrucci et al. 2013) and Go (Silver et al. 2016), historical mile markers of this field's progress as game playing agents have surpassed human grandmaster performance. Recently, video games have garnered the attention of researchers. Learning to play Atari 2600 games (Mnih et al. 2013) and Doom (Lample and Chaplot 2017) from pixels and controllers represents the latest efforts to build super-human game playing agents.

While game playing receives a lot of attention, the games themselves have been the subject of research as well, historically focusing on non-player character (NPC) behaviors, player experience modeling, procedural content generation, and massive scale data mining (Yannakakis 2012). AI academics and game engineers once found themselves separated by a sophistication gap: high-potential academic techniques on one side, and scripting (and other scalable, practical engineering techniques) on the other. This gap, however, is slowly shrinking, with some sides arguing that NPC AI is almost solved (Nareyek 2007). Player experience modeling employs AI techniques to capture predictive models of player behaviors in a variety of situations. These behaviors include in-game purchases, game mode changes, and skill scores. Procedural content generation attempts to alleviate the enormous cost and effort associated with feeding the *content furnace* as well as providing a tailored experience for the player (Togelius et al. 2011). Game data mining is critical to large game operations, providing insight into the game ecosystem. For instance, understanding how players spend their time in a game is important for tuning the current game and designing the next.

Another important area of AI research is decision support and autonomy. Decision support systems (DSS) are used by management, operations, and planners to make decisions in unstructured or semistructured environments. Expert systems, underlying knowledge-driven DSS, have been a core AI task for nearly as long as gaming (Jackson 1998). It can, however, be very simple to turn a decision support tool into an autonomous agent (Fox and Khan 2014).

The intersection of these disparate AI fields is the subject of this paper. The interactive entertainment industry, or more succinctly, game companies, employ a wide range of AI knowledge and technologies to deliver their products. Animators, game engineers, and data scientists rely on constraint satisfaction, path finding, machine learning, and many other traditional AI techniques. Game companies, like most other industries, define task-specific work roles. Animators work with other animators who are attached to teams tasked with delivering a game. Often, the only time animators, engineers, and scientists find themselves in the same room is for the company all-hands.

Imagine what happens when a highly virulent technology meme infects a cross section of game company employees. As the meme takes hold in a team, they are compelled to race after tools that promise solutions to long standing problems. Take a company with hundreds of such teams each trying to implement, deploy, and maintain their own meme-inspired solutions. Tool proliferation leads to problems with interoperability as well as knowledge acquisition and transfer.

Today, the AI and deep learning memes have spread throughout many companies, including Electronic Arts (EA). The focus is on the wonderful promises of individual

applications (e.g. conversational interfaces enabled by deep learning), neglecting the ecosystem in which they are supposed to operate. This is a minor issue for startups focused on pushing a single product out the door, but can be a terrible resource drain on larger organizations with teams spread across the world.

This paper describes our technological, as well as educational and organizational, response to this situation. We will first provide some insight into how game companies utilize AI. We then introduce our realization that we can leverage AI techniques to improve many aspects of our business. We then describe a system capable of providing AI artifacts anywhere in the organization. Finally, we discuss how this centralization has impacted EA.

Context

EA is a game publishing company with many semi-independent studios that releases about ten games per year. Given that it takes about three years to produce a AAA game, it means that there are at least thirty active development teams at any given time. Some of these teams continue developing additional content after release as well as providing operations support. Added to these game teams, many player-facing groups (e.g. customer support) and central services (e.g. marketing and publishing) also exist within EA. This situation engenders the formation of silos that limit communication and knowledge transfer.

In 2011, EA recognized that change was necessary. With thirty semi-independent studios, game teams were operating their own identity, commerce, and data warehousing solutions. Independent identity services for each game was not going to scale as EA transformed from producing ship-and-forget games to providing 24/7 high availability gaming platforms. In this context EA created the Digital Platform (EADP) to centralize these critical and common services. Maintaining multiple login services is a headache; maintaining a hundred is a security nightmare (Vacca 2009). Consolidation enables operating at scale and has improved quality of service (compare the disastrous SimCity release (Wikipedia 2017) to the relatively smooth Battlefield 1 release).

User Personas

In 2017, the phrases "AI" and "deep learning" have penetrated the common culture (Kennedy and Mifsud 2017). People have conversations *with*, rather than on, their phones and argue the legal ramifications of self-driving cars while robots pour their cappuccinos. Despite the increasing ubiquity of AI, few agree what it means. For our purposes, we view AI as a spectrum of goal-driven behavior. AI is not just a temporally defined wavefront focused on the next ground breaking application "twenty minutes in the future"[1]. Useful AI can range from amazing to mundane, and thanks to the magic of long tails the latter will out number the former. We operationalize this definition as anything deriving from "Russell and Norvig" (Russell and Norvig 2010) or any introductory textbook. Thus, everything from thermostats to the Terminator fall within our mandate.

[1] According to Max Headroom.

	Wide	**Deep**	**Broad**
	Resources	Effort	Projects
	Data Sets	*Game AI*	*Applications*
Low	Iris	Connect-Four	One
Mid	MNIST	Checkers	Hundred
High	ImageNet	Go	Thousands

Table 1: Three different types of platform scaling.

Diverse groups of people employ AI techniques and technologies in their day-to-day operations. Analysts and data scientists consolidate game data into reports and insights. Beyond machine learning, others face standard operations research optimization problems or resort to simulations for hypothesis testing. Game engineers construct behavior trees (Colledanchise and Ögren 2017) to describe NPC behaviors. NPCs must also intelligently navigate their environments, hence path finding is crucial. Animators must figure out how to stitch together short animations to form continuous sequences that give the appearance of smooth movement. They construct movement manifolds to control limbs and facial expressions (Elgammal and Lee 2008). Consequently artists, engineers, marketers, scientists, and management deal with a wide range of AI problems.

Scaling Strategy

Systems can scale in many dimensions. For our purposes, we will focus on three axes: width, depth, and breadth. Scaling for width entails acquiring more resources to achieve your goals. Bigger data sets, larger memory, more processing nodes reflect a change in scale along this axis. We reserve the dimension of depth to capture effort. An afternoon proof of concept project is shallow compared to the effort required to develop a self-driving car. Recent deep learning advances have pushed along the width and depth axes by training complex neural network architectures on massive data sets (Goodfellow, Bengio, and Courville 2016). Finally, breadth captures the number of projects. Does your company focus on one flagship project or does it sprinkle AI everywhere within the organization? In this paper, we focus on the latter; how AI can transform from handcrafted art to wholesale engineering across the organization.

Table 1 outlines the three scaling dimensions. For the dimension of width, we look at data sets. As the complexity of the data set increases from simple (e.g. $\sim$400 measurments in the Iris set (Fisher 1936)), to midsized (e.g. the MNIST database of tens of thousands of handwritten digits (LeCun et al. 1998)), to very large (e.g ImageNet's millions of hand-annotated images (Deng et al. 2009)), it becomes increasingly difficult to manage training and evaluation of models exposed to these data sets. Similar escalation can be seen in the depth dimension regarding the amount of effort. Developing a Connect-Four AI player is often an introductory AI homework assignment. Schaeffer led a small, four person, team to create a world-class checkers player (Schaeffer 2008). The DeepMind team that developed AlphaGo, the first program to defeat a Go world champion, numbered at

least twenty (he number of authors on the Nature paper) (Silver et al. 2016). Increased effort, as measured by the number of people working on a single task, requires strong organizational structure to facilitate and support team-wide communication. The final dimension, breadth, draws attention to the number of tasks themselves. The design considerations for a platform capable of supporting a few tasks are different from supporting thousands.

We have adopted several key strategies to aid this transition. First, we prefer to think about *systems over models*. Machine learning models, or any AI artifact, for that matter, capture functionality, but do not interact with the world themselves. In order to have impact, models must be embedded in systems. For example, a thermometer provides useful information about temperature, but does not do anything until it is embedded within a thermostat. This distinction is important because we choose *impact over accuracy*. Increasing the accuracy of the thermometer from two significant digits to five will not improve the performance of the thermostat. As these systems are rarely deployed in stationary environments, it is important to identify repeatable processes for constructing AI artifacts. Or more succinctly, we prefer *process over solution*. One-off solutions are fine for demos, but nobody wants to revisit their churn model every other week. Finally, in order to support these strategies, there must be a single framework to rule them all.

Components

Universal availability of AI artifacts demands a solid foundation. We have been slowly growing this foundation through a series of incremental projects over the last five years. In this section, we describe the key components that comprise our AI infrastructure. Keep in mind that each part of the infrastructure came into existence for its own reasons. There never was a long-standing plan to provide an AI service. Rather, as our small group developed real solutions to game team problems, we made sure that the design of each component was forward looking well beyond the immediate use cases. In retrospect, we were following Gall's law (Gall 1975):

> A complex system that works is invariably found to have evolved from a simple system that worked. The inverse proposition also appears to be true: A complex system designed from scratch never works and cannot be made to work. You have to start over, beginning with a working simple system.

Evolutionary design was a key strategy for the Intelligent Systems team. The team started as one machine learning engineer and grew to eleven scientists and engineers after five years. Embedded within the much larger Data Platform ($\sim$100 engineers), it had little direct influence on production architecture matters. All of our AI components entered production riding on simple use cases (e.g. delivering most played map-mode to Battlefield 1) providing the most generic solutions that simplified implementation.

A high-level view of the AI architecture appears in Figure 1. The data warehouse on the bottom of the figure serves as the foundation. It contains our repository of anonymized

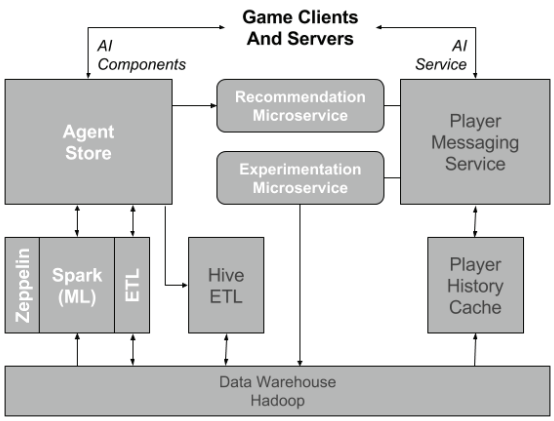

Figure 1: Block diagram of AI system components embedded within the data platform. Models generated within Spark (or other sources) are kept in the Agent Store. Agents are deployed in the recommendation engine (as a service endpoint) or externally in game clients or servers. Developers can create models in Spark through Zeppelin. The data warehouse, messaging service, and history cache were pre-existing infrastructure (black text).

player data collected via telemetry from game clients and servers. The warehouse is built using the Hadoop distributed processing framework. Machine learning takes place on our Spark platform as well as *ad hoc* analytics (through a Zeppelin notebook interface) and some extract, transform, load (ETL) jobs. An ETL job within a data warehouse generates tables from other tables. The majority of ETL jobs rely on Hive to perform necessary aggregations within the data warehouse. The Agent Store collects trained models and pipelines from Spark, and other sources (not shown). Since the storage format is Turing-complete, the Agent Store is a *code* repository more than a parameter repository. External entities, game clients, and servers can access stored models through the recommendation microservice. The player messaging system acts as a broker for the recommendation engine by combining information from the request with historical information about the player from the player history cache. Last, but most important, is the experimentation microservice that helps us ensure that changes to our AI systems have real impact. We now examine each component in detail.

Data Warehouse

The key to providing an intelligent, player first, environment is data. Game clients and servers provide a distributed computing environment within which our games operate. Clients run on many platforms, including consoles, mobile devices, personal computers, and web browsers, providing multiple sources of event data. Game servers come in different flavors as well—each backend developed by a different game team with differing priorities. A standard telemetry taxonomy is in place to unify the collection, aggregation, and storage of

player information from all of these sources within a central warehouse. The data warehouse is responsible for storing and curating player history.

Originally, the primary mission of the data platform team was to support game analytics across EA. This mission was satisfied with a standard Hadoop cluster where nearly all ETLs were specified as Hive queries. The current production cluster supporting ingestion, ETLs, and ad hoc querying is spread over 2000 computing nodes. Data platform also supported a player messaging service for delivering text and image in-game messages. It accepts requests from game clients and servers and returns a static message payload depending upon the user-defined player segments (e.g. a marketer wants to present an offer to 18-29 year old males in North America who play Battlefield 1). The messaging service employs a player history cache that surfaces relevant player information (e.g. session days for each game, kill/death ratios, etc.) for segmentation. This cache, stored in an open-source NoSQL document-ordered database (Couchbase (Brown 2012)), reduces response latency by storing a subset of features of active players.

Agent Store

Data without process is like fuel without an engine—potential but no action. The Agent Store allows us to store AI processes for later use. Originally called the "model store," it was tasked with capturing the results of our machine learning efforts. One major problem within EA was deploying machine learning models to production environments. Often a data scientist would collect model parameters and pass them to an engineer to instantiate the model in an ETL, a streaming filter, or RESTful API. This process is wrought with risk—miscommunication, library differences, and unspoken assumptions could turn implementing a simple linear model into a quarter-long engineering effort. With the introduction of the model store, we elevated machine learning models to become first-class citizens within the data warehouse. Models are stored as data that can be accessed anywhere within the company. Most importantly, models are stored as code, not parameters. Parameter-only storage ignores data transformations on inputs and outputs that are crucial to deployment (e.g. z-score transformations and one-hot encodings) and are often miscommunicated during the engineering effort.

The choice of storage format is crucial. It must be able to capture all foreseeable models as well as any data transformations. We use Portable Format for Analytics (PFA) as the model representation language (Pivarski, Bennett, and Grossman 2016). PFA was designed to represent scoring engines, and does so by representing an abstract syntax tree in JSON or YAML. The example PFA function in Figure 2 sums the squares of the first n integers.

PFA is Turing complete. Labeling it as a scoring engine representation forgoes many other applications of the processes it can describe. For this reason, we started referring to this service as the Agent Store, focusing on the general applicability of its contents.

The Agent Store supported versioning from the start, as one would expect from any other source code repository.

```
input: int
output: int
action:
  - let: {n: input}
  - let: {sum: 0}
  - while: {"<": [0, n]}
    do:
      - set: {sum: {+: [sum,
                        {*: [n, n]}]}}
      - set: {n: {-: [n, 1]}}
  - sum
```

Figure 2: An example of Portable Format for Analytics (PFA) written in YAML. This function sums the squares of the first n integers.

Most importantly, a rich collection of metadata are stored with the models to identify who, what, when, and how the model came into existence. All of this source information makes it easy to automate model evaluation and retraining. For instance, a periodic process could test for system drift by requesting a model and running it against new player data, triggering retraining if the output metrics are unusual.

Agent Creation

We think of Agent Store representation as "generate once, run everywhere". Models are rarely written by hand or custom script. Rather, we have augmented existing tools and written several new tools for generating models for deployment anywhere within EA. We have offered Spark as the central machine learning support service, focusing on SparkML and TensorFlow as our machine learning libraries of choice. Other libraries can be used, however our group only supports automatic uploads and downloads of models from the Agent Store. We have also integrated Agent Store with EA's development tools for its main game development platform used across all studios. There is also a behavior store for collecting training sets of NPC behaviors, thus offering a declarative mechanism for describing desired actions as opposed to the standard procedural approach (AI engineers specifying behavior trees by hand). We also encourage dual control of in-game elements. For instance, a designer should be able to temporarily switch an NPC to human control in order to easily specify what it should be doing under the current circumstances. Our notion of spectrum also applies to agent creation. The motivation behind many game playing agent projects is to find the utility maximizing agent representation. We have many applications for which the intermediate representations are just as useful. For instance, dynamic difficulty adjustment could drive NPC model selection to personalize interactions to maximize retention. As agents evolve through training, they may exhibit different playing styles and be useful for exploring those styles in simulation.

Agent Instantiation

The Agent Store is more than just a database of machine learning models and metadata. It supports the second

half of our "generate once run everywhere" strategy. PFA comes with interpreters in several different programming languages. Thus, instantiation points can request an agent description and evaluate it on the spot with local data. This use case is the standard one for PFA. Given that agents are abstract syntax trees, it is possible to compile the representation to any target language. The Agent Store can not only provide PFA code for interpretation, but raw code in Python, C++, or C#, as well as DLLs and JARs. These methods enable on-the-fly agent changes within running systems such as game clients and servers.

One key application that uses agents is our recommendation engine, Reco. It offers a RESTful API for providing dynamic content generated from player history (from the data warehouse) and current context (from the requester). Reco uses models to determine which actions should be taken (segmentation) as well as what outputs should be generated (computation). Our notion of recommendation is broader than the typical recommendation system definition, as the engine provides the next best activity for a player given their context. While we provide standard collaborative filtering and multi-armed bandit capabilities, the next best activity could be the result of a logistic regression, decision tree, or a simple database lookup (e.g. most played map and mode). Reco is viewed as the action layer for the data warehouse.

Reco is implemented as a microservice within the messaging service. The core of Reco is a prioritized rule system. A recommendation is defined as a collection of condition-action pairs with a dynamic priority. Conditions, actions, and prioritization functions are defined using either Ruby or Javascript. Both interpreters are available within the Java development environment used by the data platform team. Reco rules are stored in a MySql database and can be updated online and pushed from integration to production in minutes. This level of flexibility has dramatically decreased development times from weeks to an afternoon for services.

Burying recommendations within the messaging service was a historical accident. Messaging, at the time, was servicing requests from dozens of games in a variety of contexts. The initial recommendation use cases consisted of simple data retrieval tasks: e.g. retrieve most-played map-mode for a single game. Hence, serving up map-modes was shunted off to a microservice so not to impact the delivery of static content. The design decision to interpret rules at run-time was also contentious, as many within the organization argued against it (because interpreters are slow). After several POC evaluations, a consensus was reached to build a Java microservice that contained JRuby interpreter even though the Rails POC exhibited similar latency ($\sim$10ms) with less complexity. In retrospect, the original messaging system should have been designed to deliver dynamic messages from the start.

Experimentation

The Agent Store is tied to our experimentation platform in several ways. Foremost, experimentation runs *through* the recommendation engine. A recommendation, in the context of an experiment, is linked to an experiment identifier and condition assignment. This information is passed along with the recommendation, where the requester ties it to future telemetry events sent to the data warehouse. This seamless integration allows us to measure experiment performance across a wide range of predefined performance metrics. This capability is a vast improvement over various third-party solutions which require establishing new telemetry hooks and pre-experiment commitment to metrics. For example, presentation of the results of an A/B test for a feature that impacts retention often prompts someone to ask "How does it affect monetization?", sending some poor data scientist off to painfully reconstruct a monetization analysis that draws from two data sources (the warehouse and the third-party experimentation platform vendor).

In addition to providing core capabilities for the experimentation platform, the Agent Store residents can be the subject of tests themselves. Reco can act as a proxy for the Agent Store and select which models are sent to the requester for instantiation in the game client or server.

Finally, the experimentation component embodies our choice of impact over accuracy. Artifact updates are treated as experiments, where the key performance metrics (KPMs) can be compared after deployment. If these KPMs are below expectation, the update can be rolled back.

Impact

The framework we described above has been under development for five years and has stabilized in the last two years. During this time, the platform has enabled the deployment of several projects involving more than ten games. Each of the projects made significant impact on engagement, monetization, and development efforts. In this section, we will briefly outline some of the projects and their impacts.

One of our early projects was dynamic difficulty adjustment. Early analysis of churn rates and difficulty (measured in average number of trials to pass a level) showed high correlation. The first attempt used a churn prediction model to estimate the likelihood of a player not playing after a week to control game difficulty by restricting the range of random seeds used to generate board configurations. Some seeds resulted in easier board configuration than others (e.g. convenient location of resources and objectives). If a player was about to churn, the game would provide an easier board. This simple control loop increased engagement, as measured by games played per day and session time, by about 10%. Churn models were updated weekly and were phased into production over a couple of days using the experimentation tool. Setting up these experiments was the only manual component of the process.

Soon after this system had been deployed, we realized that game progression could be viewed as a random walk through a level-x-trial graph. By collecting churn probabilities and difficulty measurements, it was possible to maximize player flow through the graph using dynamic programming (Xue et al. 2017). This new method significantly increased both the number of rounds played and game play duration by about 7% over the previous method. Within these games, ad-based monetization increased as well. Our AI service architecture simplified the development, deployment, and evaluation of this new solution.

From a traditional recommendation system view, we have delivered two crucial projects (Wu et al. 2017). First, the home screen in Battlefield 1 provides recommendations for maps and modes, training videos, and information pages. The appearance of this content is under the control of our recommendation engine. Strategic selection of map/mode recommendation has increased seven day retention by 18%. We also provide product and video recommendations within the EA's Origin online store. Employing a multi-armed bandit mechanism has produced 12% increase in click-through rate.

We extended our recommendations to the area of matchmaking to deliver a novel approach to this perennial problem (Chen et al. 2017). Matchmaking connects players in online player-versus-player games. Common wisdom says that matches should be fair—players should face other players of the same skill level. We demonstrated that this intuitive strategy fails from an engagement perspective. A better way to match players in order to keep them playing involves examining their win/loss histories and skill levels to maximize retention over all players. The AI platform described above was critical to developing models and easing integration with EA's central matchmaking service.

Games with massive virtual economies often draw a wide range of nefarious activities. People are willing to pay real, non-virtual, money for virtual goods. Many websites, unaffiliated with the games themselves, exist for the purpose of exchanging credit card numbers for gold, coins, and levels. These sites drove the growth of gold farming in massive multiplayer online games (Keegan et al. 2011). Currency farming can negatively affect player experience through market inflation (excess currency chasing limited resources leads to price increases) or pay-to-win arms races (six months of game play circumvented by an online purchase).

Our AI services help EA combat these economic threats to player experience through the Bad Actor Detection System (BADS). BADS is a suite of machine learning models trained to identify accounts exhibiting questionable behavior, such as coin farming, high volume currency exchanges, and account take-over. The system relies on account labels generated by security teams who constantly monitor game activity looking for exploit indicators. These labels are then used to identify players exhibiting similar behaviors. BADS has been responsible for banning hundreds of thousands of accounts over the last two years with a false positive rate less than 1%.

Games are complex software systems. As such they are notoriously difficult to tune properly. The initial use case for our agent work involved game playing agents to explore several game facets. The first project focused on estimating level difficulty within our match-three games (Bejeweled Stars and Secret Life of Pets). This exploration uncovered some unexpected sources of difficulties in these games (e.g. initial board configurations that were significant outliers from the other boards at that level). The most difficult aspect of this work was playing from pixels. Developing image processing techniques to identify board state consumed more developer time than any other part of the project. It was also the most fragile—templates for one game could not be leveraged for the next.

In following work on The Sim's Mobile, we took advantage of in-game hooks to collect state and trigger actions. A game balance specialist was able to play through this game in about a day and a half. After instrumentation, the agents could play at the rate of about thousand complete games in about an hour. Again, our agents were able to identify many unexpected relationships in game progression at a much faster rate. The major roadblock we faced with this game was that our team had to install the game hooks ourselves. Even with a weeks worth of engineering after each release, the agents approach was significantly faster than manual game play. We have extended this system to a new mobile game coming out in 2018, but this time the game team has taken responsibility for the simulation hooks. Now, game balance simulations can be run immediately after a release freeze. We have been talking to quality assurance teams to take advantage of our approaches to further mechanize their testing efforts.

We have also leveraged our experience with game playing agents to build NPC agents. By connecting EA's internal tools for developing game engine applications, game engineers have used agents to control various in-game elements including NPCs, cameras, and vehicles. We are currently working with several new game teams to incorporate our AI services within their development workflows.

In addition to our external projects, the Agent Store has helped with the problem of data quality in the warehouse. We have a system for anomaly detection that scans table columns for missing, corrupt, or unusual data. The majority of detection relies on a very dense collection of descriptive statistics over the last twenty days (e.g. mean, minimum, and maximum of a data column with respect to a title and platform). Alerts are generated if today's statistics are out of line with recent history. For selected key metrics, we construct expectations (models) that capture seasonalities and other impactful events. These models are pushed to the Agent Store and are then instantiated in an ETL that verifies that these key metrics are moving in the right direction. The anomaly detection system has dramatically increased the quality of the data within the warehouse. Missing data is now measured in hours rather than weeks. Fewer ETLs are rerun because data problems are identified at the end of the day and often resolved within.

The central AI service with EA has had tremendous impact on the company. Foremost, we have been able to significantly increase game engagement through dynamic difficulty adjustment and matchmaking. The anomaly detection efforts have increased data quality and reduced costs incurred by rerunning ETLs. We have improved player experience by safely banning hundreds of thousands of bad actor accounts from our games. Game balancing with agents has increased game quality with minimal overhead costs.

Other AI Service Frameworks

EA is not the only organization facing model management issues in an enterprise setting. Model management (MM) is

the problem of tracking, storing, and indexing large numbers of machine learning models. Many such systems focus on facilitating the data scientist's workflow (e.g. VisTrails (Callahan et al. 2006)). ModelDB (Vartak 2017) provides a searchable history of modeling events. ModelHub (Miao et al. 2017) focuses on workflow management by proposing a domain-specific query language for models, a versioning system for model parameter storage, and a read-optimized parameter archival storage system. While easing the model development burden is important, these systems neglect the difficult problem of executing models in production environments. In organizations that split data science and engineering, productizing models is thought of as somebody else's problem (Magnusson 2016).

Google has developed TensorFlow Extended (TFX), a platform for providing production scale access to TensorFlow (Baylor et al. 2017). They wanted to provide a single machine learning platform for many tasks that ensured production-level reliability and scalability. Shared, easy-to-use, configuration of components and tools was essential, as well as, being able to perform continuous training. TFX encompassed several components including data analysis, data transformation, data validation, trainer, model evaluation and validation, and serving. TensorFlow Serving, the deployment solution of TFX, offers remote API access to models generated within this framework.

Uber has developed Michelangelo for creating, managing, and deploying machine learning models at scale (Hermann and Balso 2017). They focused on addressing a six-step workflow: manage data; train, evaluate, and deploy models, make and monitor predictions. The components of Michelangelo include a shared feature store, a large-scale model training platform, a model repository (Cassandra) for both parameters and metadata. Models can be deployed for offline usage (containerized Spark job), through an online prediction service, or as Java library.

Many cloud computing providers offer AI services along side other more traditional products. Amazon delivers predictive analytics support through Amazon Web Services via Machine Learning Service. The Google Cloud Platform supports a number of AI capabilities, such as speech recognition, translation and image content identification. Microsoft allows users to run machine learning applications through its Distributed Machine Learning Toolkit. IBM Watson Developer Cloud provides customers empower applications with Watson Intelligence as well as exposing the Watson AI engine as an analytics service. The common theme of all these commercial offerings is the desire to support siloed efforts within an organization, but not managing many projects across an organization.

Conclusion

Our strategy over the years has been to identify as many small, yet high impact, projects as possible, in order to demonstrate the utility of our services. No moon-shot projects with promises to deliver on a time scale measured in years, but small concrete wins to ratchet confidence over time. This breadth-first strategy prioritizes many system features over others. Flexibility, ease of use, and simple deployment were key design considerations that outweighed execution speed.

The effort behind this platform was as much organizational as it was technical. EA has ten thousand employees with studios in North America, Europe and Asia. Adoption of our platform was not based on the merits of the platform, but on impact of the solutions we were able to build with it. Game teams are not interested in the tech stack. They want their problems solved with the least amount of effort and risk. Systems solve problems, not models.

Education was an important aspect as well. There would be minimal, if any, adoption, if all we did was publish an API and documentation. Key to our success has been our educational efforts. By visiting game studios and delivering classes on the common machine learning platform and AI development tools, we were able to jump start adoption. During these classes, we emphasized "facilitated development" where game teams would bring us in on projects early to help guide them to solutions.

Unlike other application areas, AI is not the core component of products within the video game industry. A wide range of AI techniques can be applied throughout EA's game ecosystem, help us to deliver on our "Player First" promise. Centralization of AI services follows in the footsteps of other mission critical services such as identity, commerce, and data warehousing. To this end, our platform enables a broad spectrum of AI agent processes anywhere. We focused on breadth over depth and width, making it easy for product managers and developer to conceive and implement AI anywhere in the company.

References

Baylor, D.; Breck, E.; Cheng, H.-T.; Fiedel, N.; Foo, C. Y.; Haque, Z.; Haykal, S.; Ispir, M.; Jain, V.; Koc, L.; Koo, C. Y.; Lew, L.; Mewald, C.; Modi, A. N.; Polyzotis, N.; Ramesh, S.; Roy, S.; Whang, S. E.; Wicke, M.; Wilkiewicz, J.; Zhang, X.; and Zinkevich, M. 2017. TFX: A TensorFlow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, 1387–1395. New York, NY, USA: ACM.

Brown, M. C. 2012. *Getting Started with Couchbase Server*. O'Reilly Media, first edition.

Callahan, S. P.; Freire, J.; Santos, E.; Scheidegger, C. E.; Silva, C. T.; and Vo, H. T. 2006. Vistrails: Visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, 745–747. New York, NY, USA: ACM.

Chen, Z.; Xue, S.; Kolen, J.; Aghdaie, N.; Zaman, K. A.; Sun, Y.; and Seif El-Nasr, M. 2017. EOMM: An engagement optimized matchmaking framework. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, 1143–1150. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee.

Colledanchise, M., and Ögren, P. 2017. Behavior trees in robotics and AI: an introduction. *CoRR* abs/1709.00084. http://arxiv.org/abs/1709.00084.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 Conference on Computer Vision and Pattern Recognition*, 248–255.

Elgammal, A., and Lee, C.-S. 2008. The role of manifold learning in human motion analysis. In *Human Motion: Understanding, Modeling, Capture, and Animation*. Springer. 25–56.

Ferrucci, D.; Levas, A.; Bagchi, S.; Gondek, D.; and Mueller, E. T. 2013. Watson: Beyond Jeopardy! *Artificial Intelligence* 199:93 – 105.

Fisher, R. A. 1936. The use of multiple measurements in taxonomic problems. *Annual Eugenics* 1(Part II):179–188.

Fox, J., and Khan, O. 2014. From decision support systems to autonomous agents: How can we ensure ethical practice? In *AISB 2014 - 50th Annual Convention of the AISB*.

Gall, J. 1975. *General Systemantics*. Ann Arbor, Michigan: General Semantics Press.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Hermann, J., and Balso, M. D. 2017. Meet Michelangelo: Uber's machine learning platform. https://eng.uber.com/michelangelo/ [Online; accessed 06-November-2017].

Jackson, P. 1998. *Introduction to Expert Systems*. Addison Wesley, third edition.

Keegan, B.; Ahmad, M. A.; Williams, D.; Srivastava, J.; and Contractor, N. S. 2011. What can gold farmers teach us about criminal networks? *ACM Crossroads* 17(3):11–15.

Kennedy, K., and Mifsud, C., eds. 2017. *Artificial Intelligence: The Future of Mankind*. New York, NY: Time Inc. Books.

Lample, G., and Chaplot, D. S. 2017. Playing FPS games with deep reinforcement learning. In Singh, S. P., and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 2140–2146. AAAI Press.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, 2278–2324.

Magnusson, J. 2016. Engineers shouldn't write ETL: A guide to building a high functioning data science department. http://multithreaded.stitchfix.com/blog/2016/03/16/engineers-shouldnt-write-etl/ [Online; accessed 10-November-2017].

Miao, H.; Li, A.; Davis, L. S.; and Deshpande, A. 2017. ModelHub: Deep learning lifecycle management. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 1393–1394.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with deep reinforcement learning. *CoRR* abs/1312.5602. http://arxiv.org/abs/1312.5602.

Nareyek, A. 2007. Game AI is dead. long live game AI! *IEEE Intelligent Systems* 22:9–11.

Pivarski, J.; Bennett, C.; and Grossman, R. L. 2016. Deploying analytics with the portable format for analytics (PFA). In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 579–588. New York, NY, USA: ACM.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice Hall, third edition.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* 3(3):210–229.

Schaeffer, J. 2008. *One Jump Ahead: Computer Perfection at Checkers*. Springer Publishing Company, Incorporated, 2nd edition.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Trans. Comput. Intellig. and AI in Games* 3(3):172–186.

Turing, A. M. 1953. Digital computers applied to games. In Bowden, B. B. V., ed., *Faster than thought: a symposium on digital computing machines*, 288–295. London, UK: Pitman.

Vacca, J. R. 2009. *Computer and Information Security Handbook*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Vartak, M. 2017. MODELDB: A system for machine learning model management. In *8th Biennial Conference on Innovative Data Systems Research (CIDR 17)*.

Wikipedia. 2017. SimCity (2013 video game) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=SimCity_(2013_video_game)\&oldid=797741532 [Online; accessed 11-September-2017].

Wu, M.; Kolen, J.; Aghdaie, N.; and Zaman, K. A. 2017. Recommendation applications and systems at electronic arts. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, 338–338. New York, NY, USA: ACM.

Xue, S.; Wu, M.; Kolen, J.; Aghdaie, N.; and Zaman, K. A. 2017. Dynamic difficulty adjustment for maximized engagement in digital games. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, 465–471. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee.

Yannakakis, G. N. 2012. Game AI revisited. In *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, 285–292. New York, NY, USA: ACM.