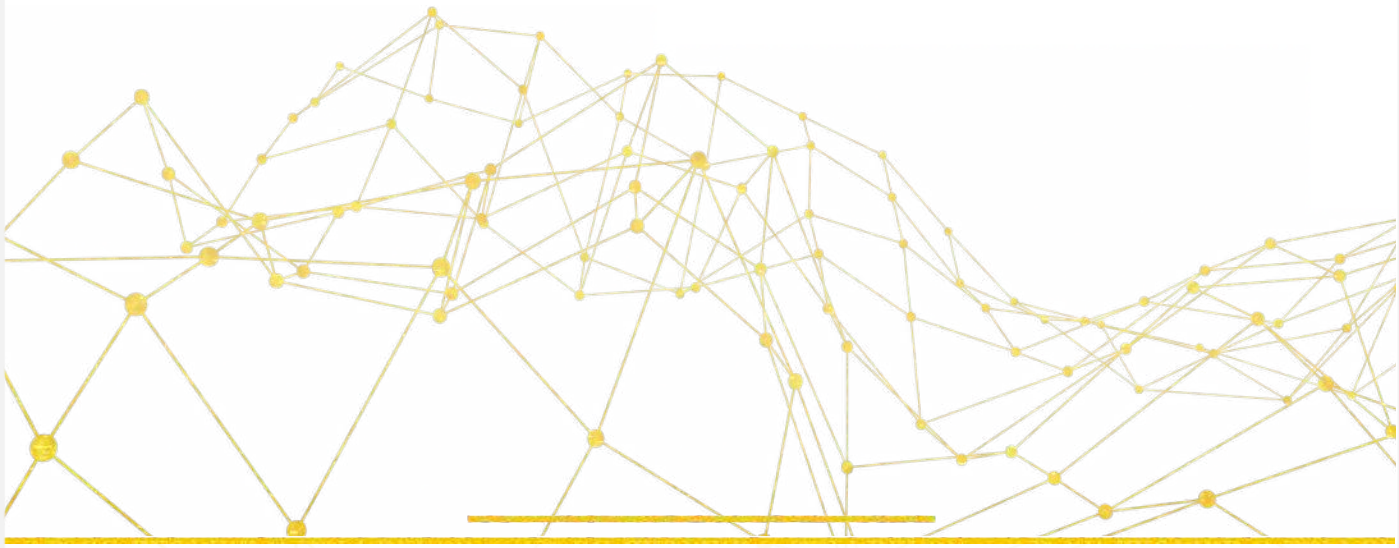
HOCHSCHULE DER MEDIEN

Whitepaper on

Player Imitation

How to make your players redundant



By Linus Griebsch Matrikel Nr: 40839,

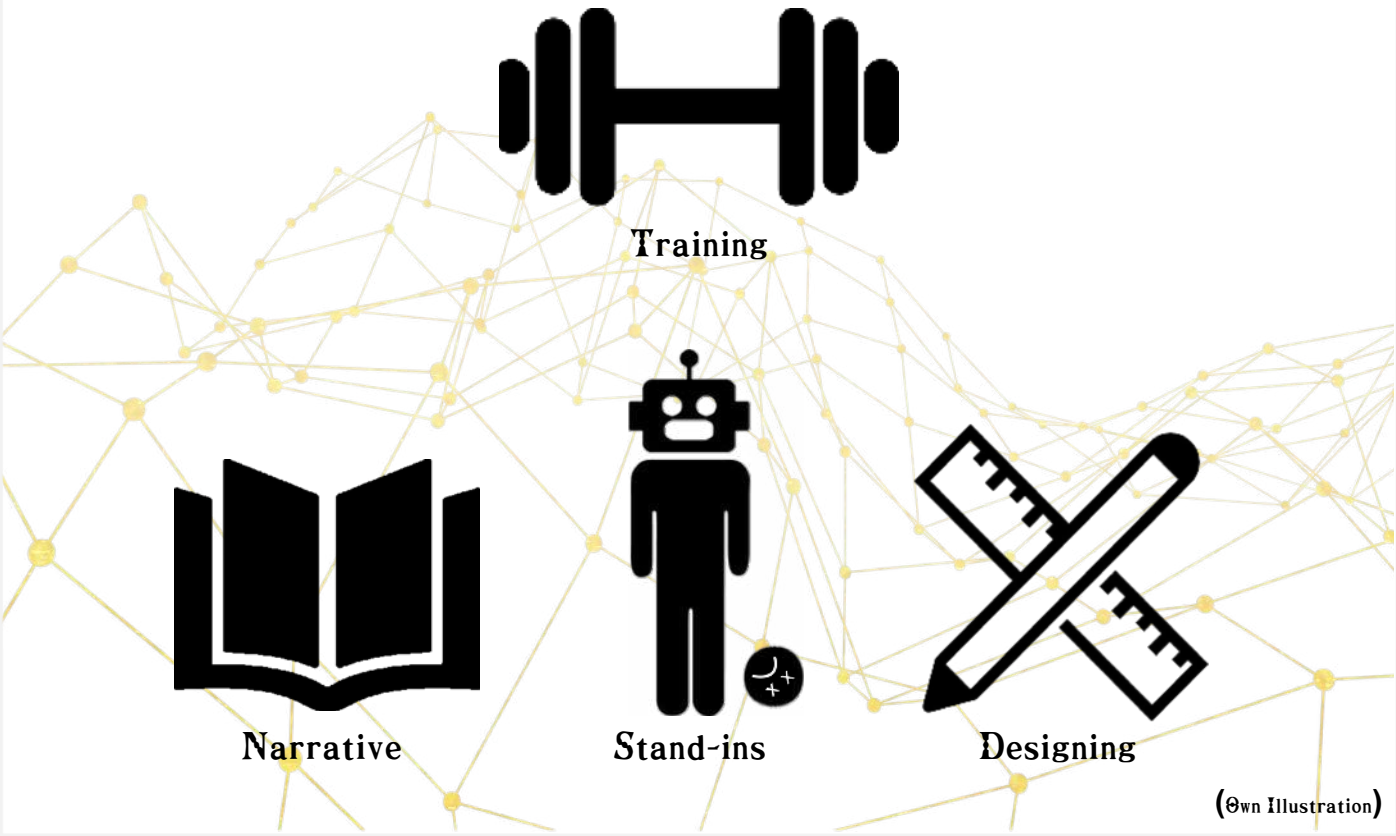
Current Topics - Aritifical Intelligence Summer Semester 22

The Concept behind Player Immitation

Within the context of video games and AI, player imitation refers to an AI agent that imitates the behavior of a player. In an abstract sense, the goal of the AI is to play "like" the player. This is too vague of a goal for an AI to work with however and therefore a more exact formulation is:
Given a situation, the AI agent should enact the same actions as a player would in the same situation. This includes mistakes players make or patterns/strategies players employ. Looking at this from a machine learning lens, all we really are trying to do is predict player actions and use the prediction as input for an AI agent. Lucky for us, prediction is a common usage field of machine learning research, and something AI excels at.

Application

It might not be immediatly self evident why bots that play like clones of humans have any merit, so lets look at some of the major benefits. First a study by Soni et al.[1] has shown that human-like bots are more interesting to play with. This benefits imitation bots too, as a bot perfectly imitating a human should be pretty humanlike. Additionally imitation AI enables utility in the following aspects of video games: Player training, AI behaviour designing, AI Stand-ins, and Narrative benefits.
The AI system allows players or game developers to engage with game AI in new ways which can drastically improve the play experience or game design process.



(Own Illustration)

Player Training

Within competitive games, especially adversarial games where players play against other players, having human-like AI is a huge benefit. This is because humans have some behavioral quirks that are hard for conventional AI methods to replicate, such as reaction times and predictable yet adaptable behavior patterns.
Due to this normal AI opponents are often insufficient as sparing partners in preparation to play against real humans. The strategies to beat a normal AI largely boils down to exploiting its set-in-stone behavior patterns, while beating a human-like agent requires far wider mastery over the game. But this is only the downside of common AI approaches, on top of negating these downsides immitation based AI also has a decent amount of benefits unique to it:

Community-powered

The AI system can be set up to automatically learn from any player playing the game, and create an AI replica of that player, which will be stored in the cloud. The benefit of this is a huge library of AIs of different skill levels and playstyles available to both the developer to design content with and for other players to play against. Players can then use these AI opponents for ego-safe offline practice, which has the benefits of infinite replayability, since the AI opponent is endlessly available for training, unlike a real human.

AI Champions

Most competitive games will have famous players known for their skills and success in competitive settings like tournaments. The average player will realistically never get to practice with these champions, but AI clones of them could be available for every single person who wishes to learn from them.

Meta adaptation

AIs that keep learning from players during a game's lifetime will keep learning new techniques and strategies as the game ages, without the need of the developer to adapt the AI at all. The benefit of this is that AI practice partners will never be outdated.

(1)

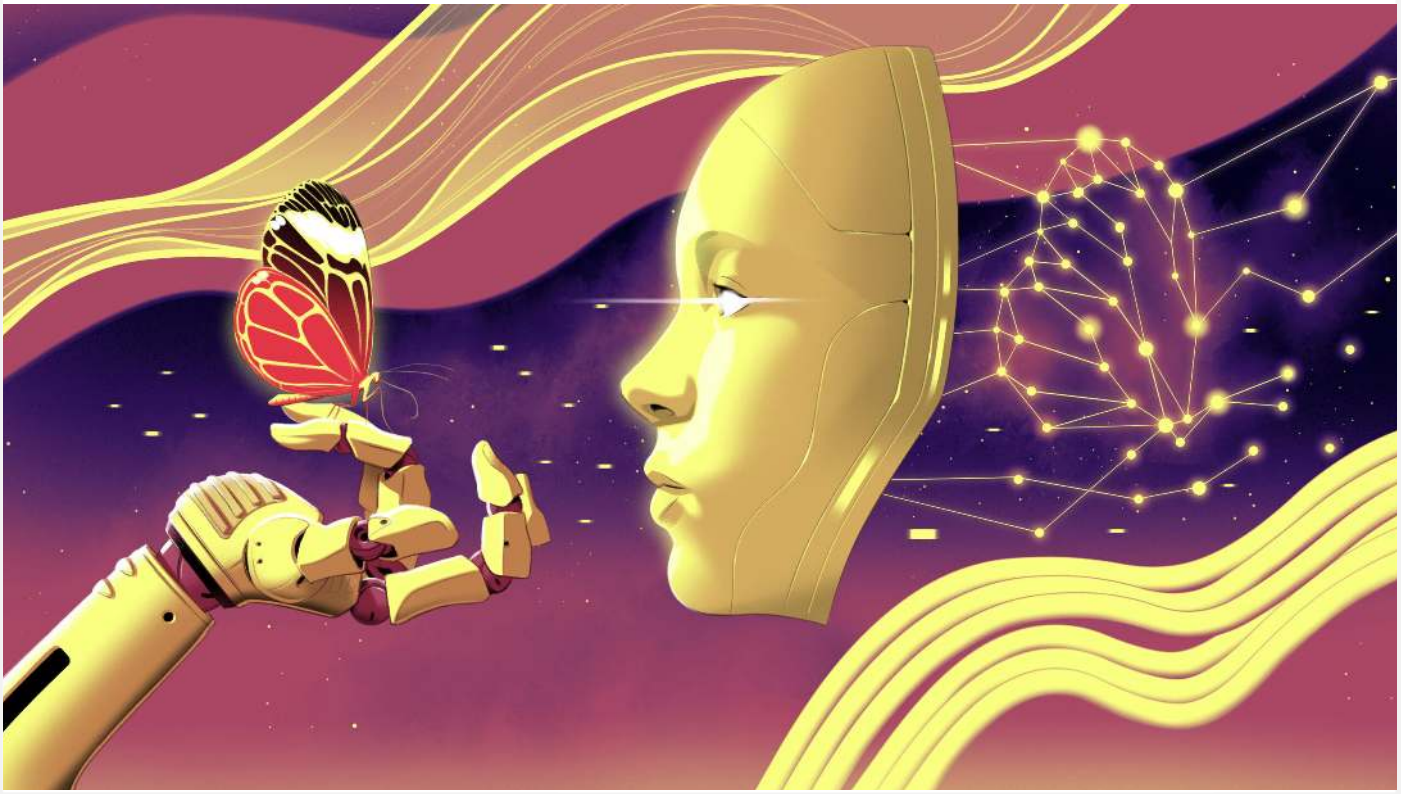
Self-reflection

Not only can players play against copies of other players, but they can also play against copies of themselves. The benefit of this might not be immediately obvious, but this allows players to reflect on their own mistakes or bad behavior patterns unlike anything else. Anytime the AI makes a mistake that the player can exploit, that mistake is a reflection of the behavior the player engages in. And if the player learns to exploit that behavior, the AI will learn to exploit it too. This in return forces the player to adapt and fix the exploitable behavior. Due to this feedback loop, the AI system should always be at the same skill level of the player, giving them the opportunity to improve.

Stand-ins

Team games that rely on cooperation between players can benefit from AI stand-ins. In these games, two or more players usually form a team that aims to win the game together. Commonly these games are played online over the internet, which leads to problems when one or more players in the team lose their connection. Usually, the game is then paused until the disconnected player reconnects because the loss of one player even for a short duration can set a team back significantly in competitive games. This leads to frequent waiting times in multiplayer games.

An alternative solution would be stand-in AI for the player that disconnected. With normal AI this would result in an unfair advantage or disadvantage to the team with the AI, as the disconnected player might be a worse or better player than the AI. An imitation AI should be about as good as the disconnected player however and also copy the player's play patterns, which lets it serve as a good temporary stand-in for a human.



(2)

Further Benefits

Demonstration

Outside of just playing against AI bots, the AI system can also be used for demonstration purposes. For example: If a player is lost in a certain game state and doesn't know what to do, or maybe they know what to do but would like to see a more optimal way of doing it, the player could have an AI bot demonstrate what a highly skilled player would do in the same situation. Especially coupled with replay systems this could be a potent tool for learning. A player wishing to know how they could have played better might watch a replay of them losing a match, pause the replay and see what a better player might have done in a given situation. Coupled with the AI system being meta-adapting, this could essentially function as an adaptive advanced tutorial system that can show players the correct behavior in any given situation.

AI behaviour design

A problem with AI systems currently utilized in the gaming industry is that they requires expert knowledge to be adapted. This means that if a gaming studio wishes to have modular AI that behaves differently in different scenarios, the AI programmer needs to frequently adapt the AI system to suit the vision of the game designers. With an imitation-based AI system, any layman can create their own AI behavior simply by recording their own actions. If a designer wants an AI that constantly jumps, all they have to do is record themselves playing the game and constantly jumping. This allows game designers to at the very least prototype different AI behavior rapidly without the necessity of an AI programmer. It also opens up completly new game mechanics about teaching AIs behaviour to fulfill the players goals, like having real time strategy games where the micro controlling of units is replaced with prior training of the units AI.

(3)

Narrative

The combination of the events and activities of the gameplay aligning with the narrative the game is trying to tell is called the Ludo-Narrative. Unlike movies and other non-interactive narrative media, games benefit from combining their game mechanics with the narrative of the story or can be hurt by the clash of the gameplay with the narrative.
For instance, playing a game in which the player has the freedom to kill random NPCs while the player character is portrayed as a pacifist in the narrative will evoke what is called Ludo narrative dissonance. This dissonance affects the story usually negatively. In the example, the character of the protagonist would be hard to take seriously in their struggles for non-violent conflict resolution if the player just killed a dozen of NPCs beforehand.

The benefit of Imitation AI in relation to the narrative is then that it adds game mechanics to narrative elements that prior had little in the way of game-play representation. Namely imitation.
The scenario is not uncommon in games, fighting an opponent that is a reflection of yourself, or that learned to copy your behavior, or maybe it turned out you are just one of a thousand clones that all act just like you.
Usually, these NPCs are shown to be similar to the player by looking like the player character and having the same or a similar move set. The actual behavior patterns for these copies are usually completely different from the player character, however, causing disbelieve in the narrative that these copies are really the same as the player character. Imitation-based AI perfectly solves this issue, but that is not the only narrative aspect it can be used for.

When it becomes hard to tell humans and AI apart game designers can use this for narrative tricks.
Games that rely on simulating a large number of humanlike NPCs can use AI to fake populated worlds. Or coop games can use AI to play tricks on the player, with them not knowing if the person they are playing with is actually a human ally or an AI copy of them in disguise.

Implementations - Deep Learning

We took a look at the potential of imitation based AI, but how does it work out in practice?
Few games have actually implemented imitation AI, but was it at least successful in those that did implement it?

Samurai Shodown

Samurai Shodown or Samsho is a 1vs1 fighting game. The entire game-state of Samsho plays on a 2D plane and the goal of both players is to hit the opponent until their life depletes to 0. Fighting games like Samsho are a great environment to implement novel AI approaches, as the parameter count comprising the game state of the game is relatively low. This is because the environment in most fighting games is standardized, meaning that even if different levels are chosen, the choice is only an aesthetic difference. Additionally, character actions are also set in stone and cannot be customized by the player. This makes for a reliable and predictable environment perfect for competitive settings like tournaments where fighting games strive, but also a great environment for AI to learn.
So how did Samsho implements its imitation AI? Sadly no detailed information about the architecture of the system is available, but it is known know that their Ghost system as it is called is powered by neural network-based deep learning.
The goal of the ghost AIs was to create a clone AI of the players playing, copying their exact behavior and letting other players play against them. Sadly the ghost AIs aren't working out so great. Users have described them as broken, and indeed video footage indicates that these AIs seem to break quite easily, getting locked in a loop of repeating bad actions.
Creating imitation-based AI doesn't seem as easy as just throwing deep learning at it, even in a relatively simple game space like fighting games.

Forza's Drivatar

Forza is a racing game that distinguishes itself from other titles by basing its mechanics on physics simulations. This means that vehicle steering is heavily affected by all sorts of aspects like the weight of the car, the material of the tires, the roughness of the road, or the weather conditions of the racing track.

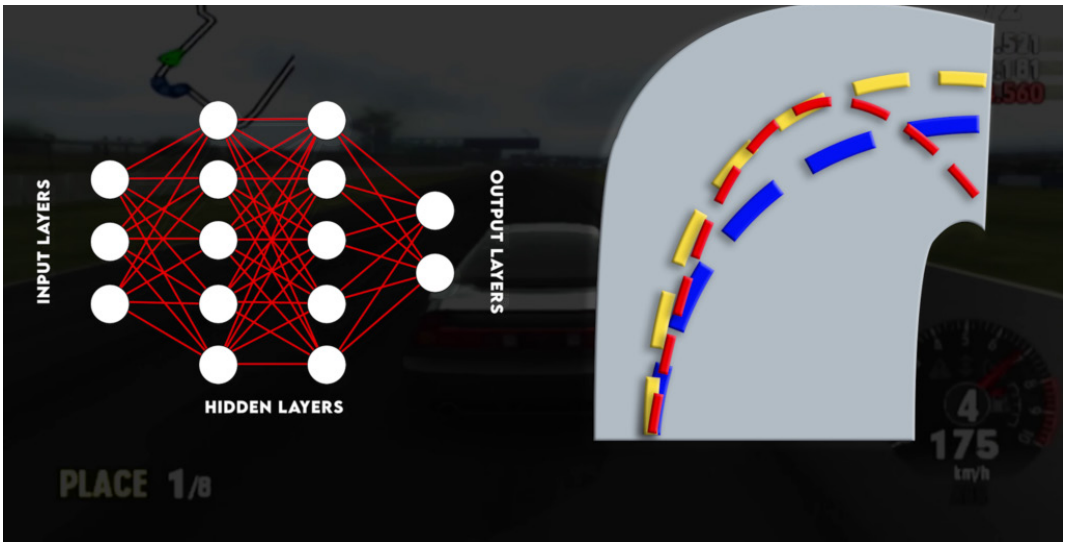
Players can customize their vehicles heavily and they also have a large number of vehicles available to choose from. Combined with the great variety of tracks available to drive on, the game state differences available in Forza seem quite large compared to that of most fighting games.
Yet Forza's Drivatar system has been working well for quite a while, and unlike Samurai Shodown there is enough information on their Drivatar system to understand how they made it work.

Like Samsho, Forza's Drivatar also uses neural networks that learn by observing player behavior. However, unlike the prior, the aim of the Drivatar AI isn't to 100% copy player behavior. Instead, the goal is to roughly model how a player would behave in any given situation. This is because the AI also needs to be able to deal with situations that the player has never experienced, like driving a car the player never used, or racing on a track the player has never driven on.

So how is the player behavior modeled?
Forza employs a separate AI system from Drivitar which models the optimal way to race on a given track, which can be seen as arrows on the racetrack. This optimal path is then employed as a baseline from which the players deviate. The job of the neural network is then to learn the deviation of the player it is trying to model and generalize this behavior for situations that are similar but not the same.

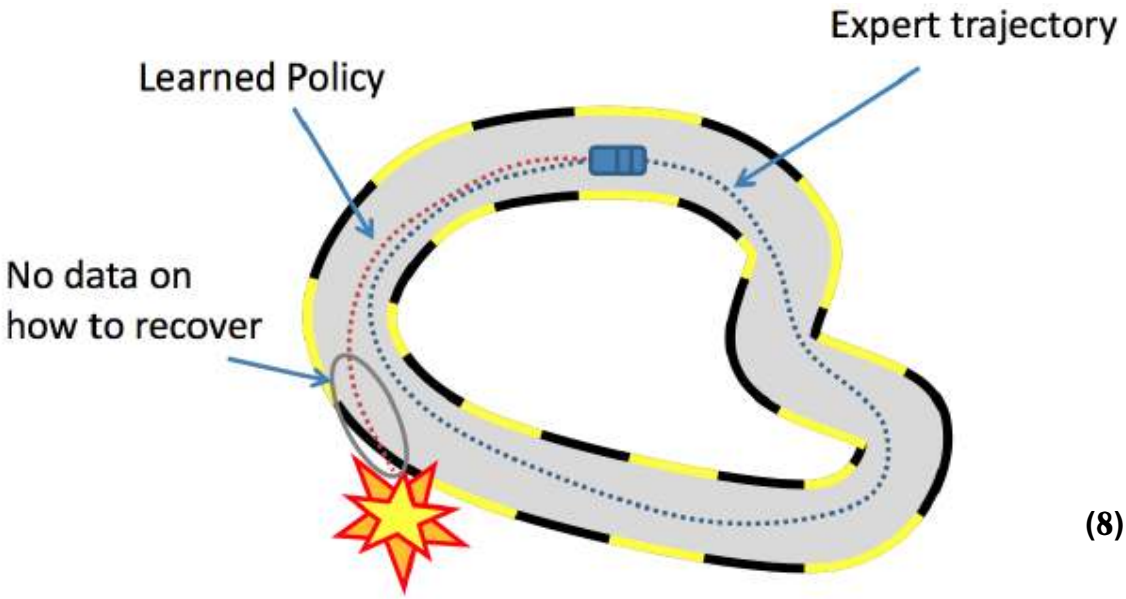
(6)

Generalization is important to avoid the AI from collapsing the moment it encounters a new situation it has not observed yet. To generalize the player behavior the developers created different categories of curves and had the neural network learn player behavior for the different categories. This then allows the AI to re-use behaviors on roads that the AI had not learned on yet. A sharp curve in any map will look relatively similar, and the neural network can use the optimal path as a baseline for adaptation. All of this effort is necessary because generalization has a major problem. Given the same input, minor differences in a game-state can result in major differences in outcome. If you took a corner very tightly in one track, the same way of taking the corner might have you crash into the wall because the track is a little narrower.



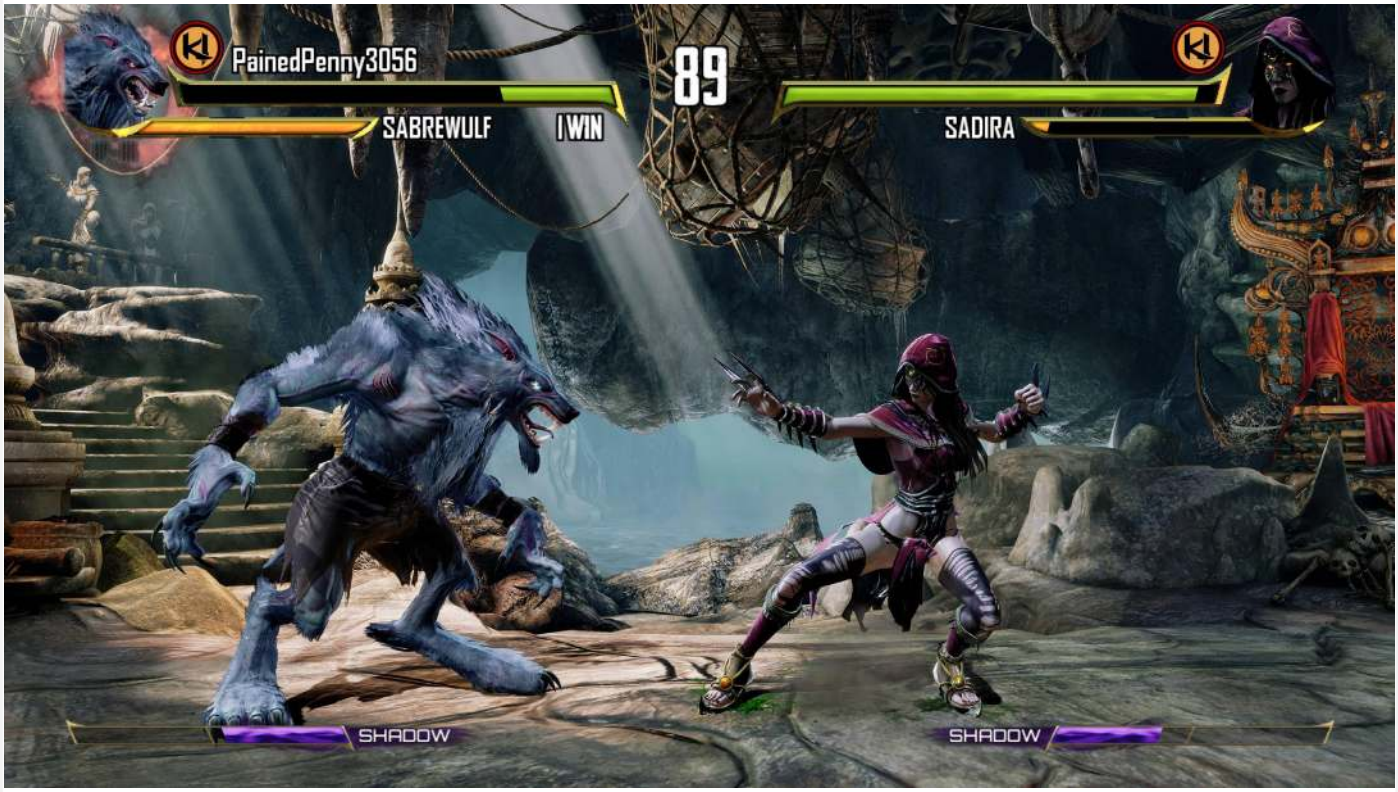
(7)

On top of this, the Drivatar system has an additional layer to adapt AI behavior. This layer exists to make sure AI opponents aren't exhibiting game-disturbing behavior, like ramming the player constantly or just drifting in circles endlessly. Drivatars also utilize rubber banding, meaning that when they fall too far behind the player the car controlled by the AI will perform better. This is to ensure that the AI opponents can catch up and the race is still tense.
Overall Forza's system does a good job of model and generalizing player behavior, but doesn't entirely imitate player behavior.



(8)

Implementations - Case-Based Reasoning

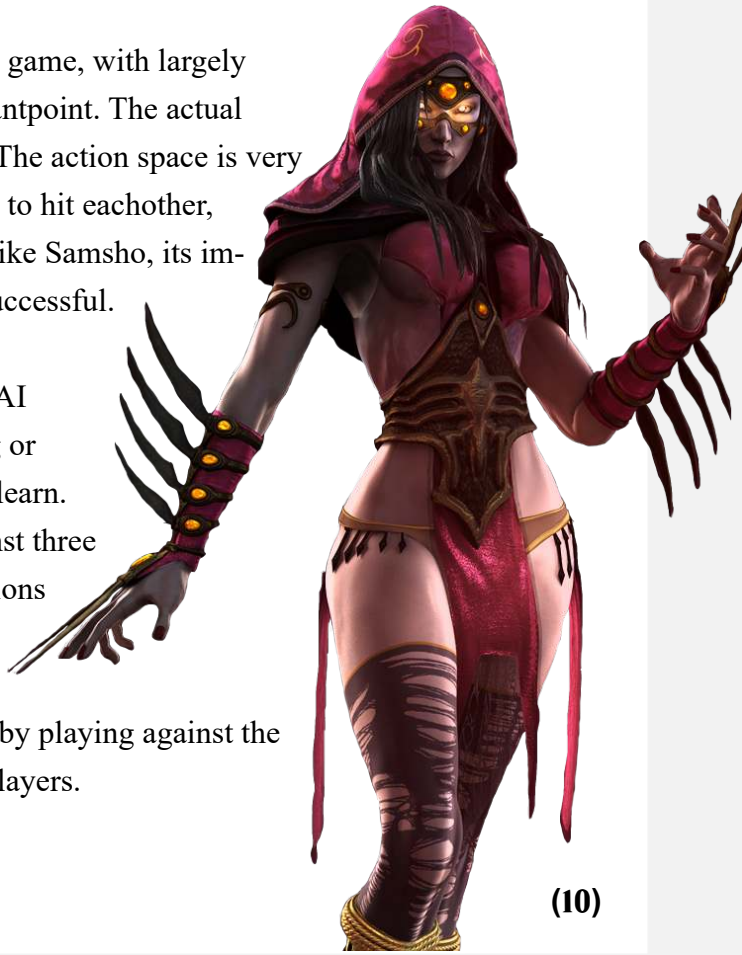


(9)

Killer Instinct

Killer Instinct, like Samurai Shodown, is a 2D fighting game, with largely similar mechanics, at least from an AI development stantpoint. The actual gameplay the player experience is noticably different. The action space is very similar to Samurai Shodown in that its 1v1, players try to hit eachother, character states and stages are predictable, etc. But unlike Samsho, its implementation of imitation AI is generally considered successful.

The difference between the two is that Killer Instincts AI system uses a technique named Case-Based Reasoning or CBR for short, which does not use neural networks to learn. Players create their AI clones initially by playing against three different AI opponents designed to test the player's actions in common game-states like how the player deals with projectiles, with up-close combat, or with jumping opponents. Afterward, the player can train their AI clone by playing against the AI clone or by playing against the AI clones of other players.



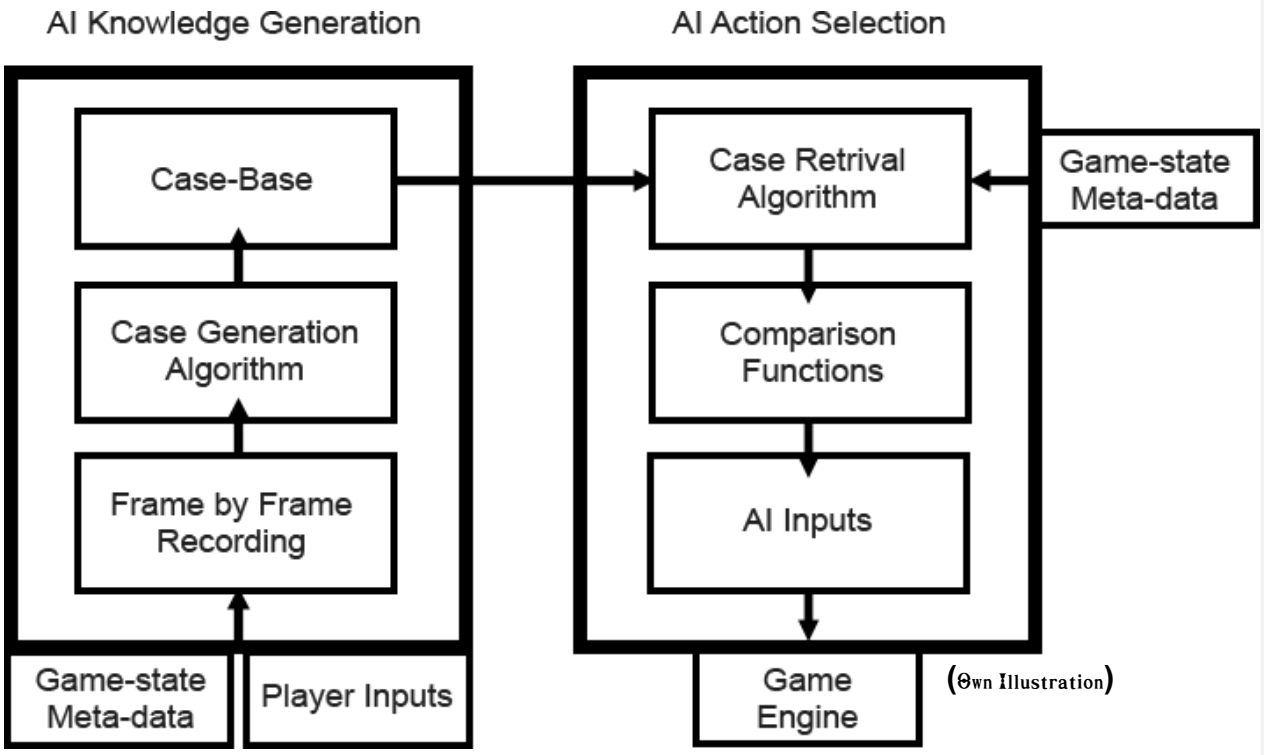
(10)

Case-Based Reasoning

Case-Based Reasoning

The concept of case-based reasoning is relatively simple in comparison to neural networks. The idea is that we save past actions in combination with the game state so that we can replay them later on in a similar game state. The visual demonstrations stem from my own implementation in Ikemen Go, but the concept is the same as Killer Instincts.
In the case of Killer Instincts, we save actions like attacking or movement and add metadata to it about the game state like the position of the characters, the status of the characters like if they are attacking, jumping, or getting hit, and any other relevant variable for decisionmaking. The same concept can be implemented purely with player inputs as well, by grouping a sequence of inputs into a case and adding metadata to it.

Once we saved enough of these cases, we can activate the imitation AI, which will search through the metadata of all of the cases to find the case that is the most similar to the current game state.
It will then keep replaying that case and following cases until the game state is no longer similar enough to the metadata within the currently playing case. At this point, the AI will search for the best case again and restart the cycle.



(Own Illustration)

Case-Based Reasoning - Generation

Before the AI can make any decisions on which actions to take it needs a collection of cases to search for actions. Therefore we need an algorithm to generate those cases, which will do in two steps. First, we will record reaply data of some gameplay and then split the recording into cases with added metadata.

Recording

The recording process can vary but generally is very similar to the recording process of replay files. Our goal is to record data that lets us re-create the entire match so we can watch it again. Commonly this is done by recording the player inputs for every frame of the game, but other methods exist as well. The main difference for CBR usage is that we want to annotate our recordings with metadata, as that will be necessary information to store the metadata for our case. What game variables we choose to save as metadata depends what variables we consider important for our AI decisionmaking.

Case Creation

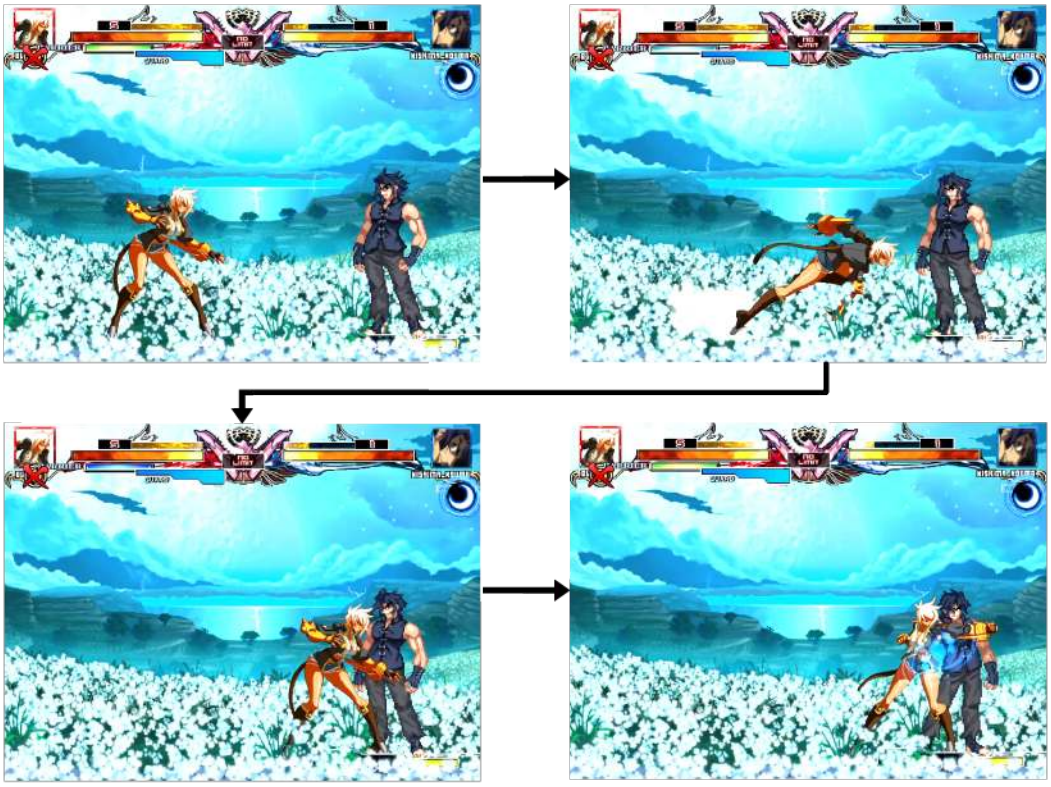
Once we have saved the recording of an entire match we need to decide how to split this replay into cases. This is necessary because saving every frame as its own case would require huge amounts of data, and also lead to a large amount of redundant cases with almost identical game-states. There are no hard rules on how to separate replays into cases but my implementation model follows these rules:

1. When a character takes an action that moves it into a new state a new case is made starting with the input.
2. When a character is forced into a new state a new case is made starting with the new state.
3. When a character moves without changing states a new case is generated every ~0.2 seconds.

The point of splitting replays like this is to make it more likely to catch the decision points of the player, while also making sure that cases aren't made too frequently to better capture sequences of actions. But the above services only as an example and the case generation algorithm should be tailored to the game it is implemented in.

When separation points in the replay have been found we need to create the data structure for the case. This can be done by saving the metadata for the case, and a pointer to the segment of the replay file that the case represents, but other approaches work as well. Finally, we save the case in a data structure, called the case base, to preserve the sequential nature of the replay. We do this to preserve chains of actions players engaged in.

Recording



Meta Data
(If this is the gamestate)

Cases Array

Dash → Punch

Actions
(Then do this action)

(Own Illustration)

Case-Based Reasoning - Replaying

Once we have enough cases for the AI to function we can now use those cases to make our AI function. To do so we need to make sure we have an algorithm to find a case, choose the best case, and then play the case back.

Case Searching

Before we can figure out which case is the best one to use we need to solve two problems:
When do we search for a new case, and how do we efficiently search for the best case?

Searching for a new case every frame results in performance issues, in addition, to constantly interrupting whatever chain of actions the AI was about to perform. It's also not very humanlike behavior. Humans don't consider their next actions 60-120 times a second.
So we need an algorithm to determine when to check for a new case. In the Ikemen implementation this is done with the following rules:

1. When a case is finished playing, search for the best case in the case base.
   Compare the best case to the next case that would naturally be played next from the array. Only if the best case is significantly better than the next case do we use the best case, else we will play the next case in the array.
2. If a drastic game state change occurs, search for the best case in the case base and play that, even if another case was currently playing.

The purpose of these rules is to ensure that sequences of actions that are saved as sequences of cases have a high likelihood to play out. But also allow the AI a frequent opportunity to switch cases if there is a far better case available. The drastic game state change rule exists to ensure that the AI doesn't have a terrible reaction time. In Ikemen, a drastic game state change would be the AI getting hit for example.

Once we know when to search we now need an algorithm that actually does the searching. The easiest variant is just looping over the entire case base, but that is also performance inefficient. A better option is utilizing a search and sort algorithm that index the case base based on its parameters and searches for the most similar case without having to look at the entire case base.

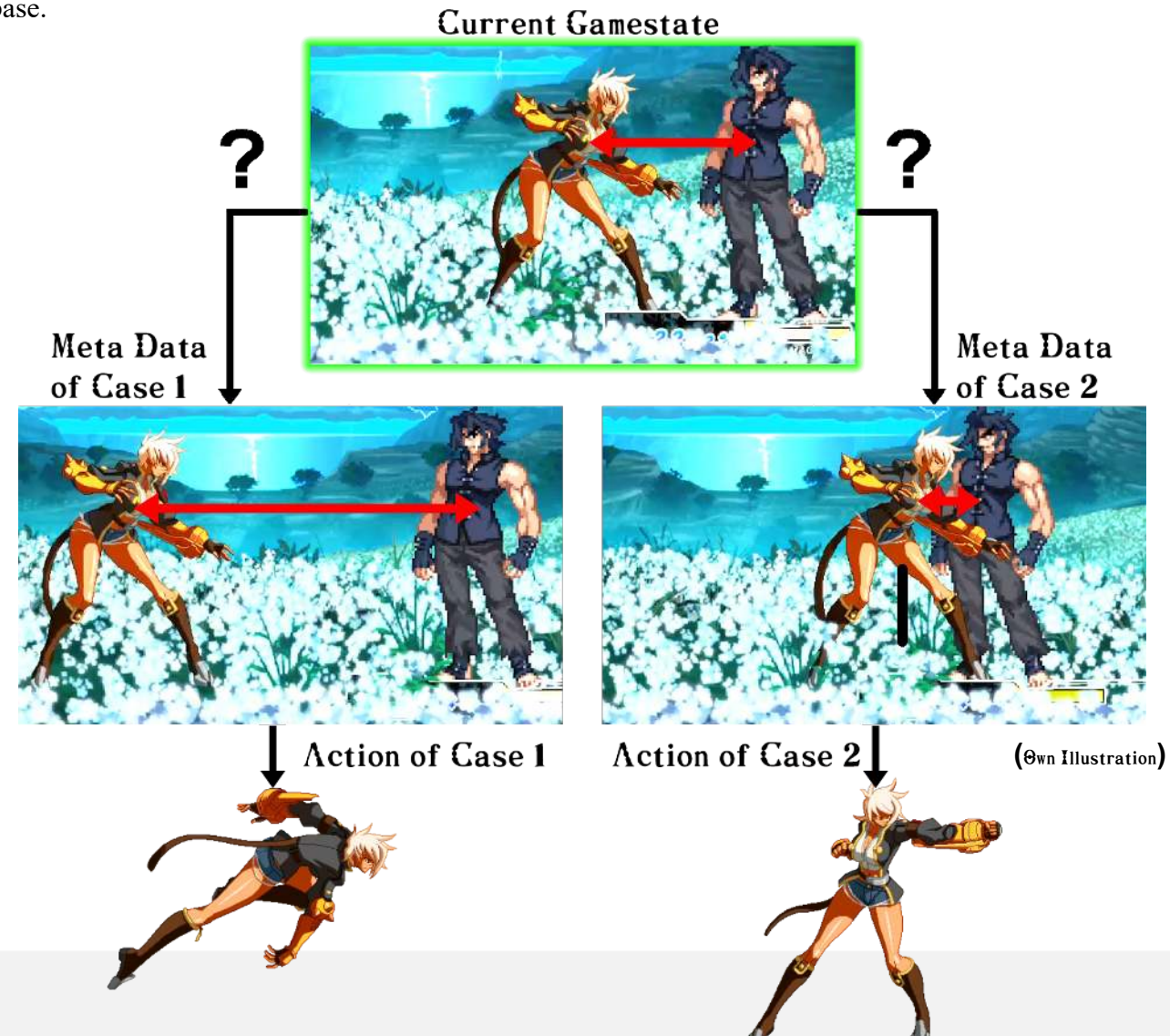
Case Selection

To evaluate whether a case is similar to the current game state we need comparison functions. These functions evaluate individual parameters and assign some value to the similarity of these parameters. For instance, the health of the player could be a parameter that we compare between the case and the game state. As an example, if the health of the player in the case is at 10% and it is at 100% in the game our comparison function would put out 0.9 as a dissimilarity score. This dissimilarity score indicates that the difference between the case is 90% dissimilar, where 100% is the maximum dissimilarity possible.

We will run comparisons like this for every single parameter and add weights to those parameters to make sure some parameters are treated as more important.
In our example, the health dissimilarity score is 0.9 and the player position dissimilarity is 0.1. The position is a more important variable so we weigh it times 5 while we weigh the health only times 1. This means that the position beeing similar to the gamestate is 5 times as important as the life. The best case at the end of this process is the case with the lowest dissimilarity score, and that will be the case we end up choosing.

Replaying

Replaying is basically the same as it is for any replay file. If we saved the player's inputs with the case we simply feed those inputs to our AI character and have the character execute them.
base.



Case-Based Reasoning - Drawbacks

While the theory behind CBR is relatively simple it does come with quite a few drawbacks compared to a deep learning approach:

1. The comparison functions by which the metadata gets compared to the current game state are usually expertly crafted. This is necessary because the evaluation of which parameter has higher priority when deciding one's course of action requires in-depth knowledge of the game. It is theroretically possible to automate the comparison function evaluation through algorithms or deep learning, however.
2. The AI data is quite sizable compared to neural network models as it is composed of annotated replay files. This also means that the more the AI learns the bigger its size will be.
3. Because the number of cases stored can become quite a lot, the AI can negatively affect the performance of the game, as searching for an optimal case takes a while.
4. Without adapting the method to better generalize it is poorly equipped to deal with new situations that it has not observed before.

A lot of these drawbacks can be mitigated or alleviated through clever adaptation of the system or by combining it with other AI systems, but even relatively barebones CBR systems can serve their purpose very well already.

Conclusion

Part of the reason why I am personally so invested in imitation AI is that its is potential, just lying on the table waiting to be grasped. It was already utilized successfully, it is not substantially more work than other AI approaches, and it opens up tons of possibilities for video games that are almost completely unexplored.
A big reason for this is that despite artificial intelligence being a hot topic garnering a lot of attention from the general public, it hasn't taken up much space in the games industry yet. Novel AI approaches have already been successfully implemented in AAA video games, but have not spread further than their initial experiments. AI systems being subpar shouldn't be the standard in modern video games, when there are more successful AI solutions that failed to catch on.

This might be partially a marketing issue, while AI is a hot topic, it's hard to convey to consumers why a special AI makes a game more entertaining. AI is hard to visualize, which is why AI-based media generators, like the art generator Dall-E 2 work best to garner interest. General audiences don't need to understand the AI process and its benefits, they just need to see the generated output to know the AI does something worthwhile.

So I advocate for anyone who desires an improvement in the medium of video games to take a look at imitation AI, play Killer Instinct or Forza to experience the potential, and spread the word. And for any video game developers that caught interest to utilize the system. The source code for my implementation in Ikemen is available here: https://github.com/KDing0/Ikemen-GO
But even without the source code, the system is not hard to re-create from the outlines given in this paper.

Sources

https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c
https://www.siliconera.com/samurai-shodowns-dojo-mode-is-pretty-useless/
https://www.gamedeveloper.com/design/how-forza-s-drivatar-actually-works
https://arstechnica.com/gaming/2020/09/war-stories-how-forza-learned-to-love-neural-nets-to-train-ai-drivers/
https://www.wired.com/2014/09/forza-horizon-2-drivatars/
https://www.topgear.com/car-news/gaming/meet-man-behind-forza-motorsport
https://web.archive.org/web/20070314015129/https://forzamotorsport.net/news/pitpassreports/pit-pass36-2.htm
https://venturebeat.com/games/how-microsofts-turn-10-fashioned-the-ai-for-cars-in-forza-motor-sport-5-interview/
https://www.theguardian.com/technology/gamesblog/2011/sep/06/forza-4-interview
https://www.eurogamer.net/digitalfoundry-the-making-of-forza-horizon
https://github.com/KDing0/Ikemen-GO

[1] Bhuman Soni and Philip Hingston. Bots trained to play like a human are more fun. pages 363 – 369, 07 2008. ISBN 978-1-4244-1820-6. doi:10.1109/IJCNN.2008.4633818.

Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun. 7, 1 (March 1994), 39–59.

Aamodt, Agnar. (2009). Case-based reasoning for improved micromanagement in Real-time strategy games. Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning, ICCBR 2009.

Video Sources

How Forza’s Drivatar Actually Works :
https://www.youtube.com/watch?v=JeYP9eyIl4E

How Forza’s Racing AI Uses Neural Networks To Evolve:
https://www.youtube.com/watch?v=XB9lf7iJbRw

Designing AI for Killer Instinct:
https://www.youtube.com/watch?v=9yydYjQ1GLg

Training the Shadow AI of Killer Instinct (2013):
https://www.youtube.com/watch?v=Etj5ykJugwU

Image Sources

(1) Samurai shodown character art:
https://sc6.soularchive.jp/character/haohmaru.php
(2) AI image :
https://www.quantamagazine.org/self-taught-ai-shows-similarities-to-how-the-brain-works-20220811/
(3) Wh40k necron army:
https://warhammer40000.com/de/
(4) Nier Automata cover art
(5) Screenshot of the game Samurai Shodown:
https://metro.co.uk/2019/06/27/samurai-shodown-review-the-spirit-of-snk-10077911/
(6) Screenshot of the game Forza:
https://www.nme.com/features/gaming-features/7-things-i-wish-i-knew-before-starting-forza-horizon-5-3089653
(7) Forza drivatar learning visualization:
https://www.gamedeveloper.com/design/how-forza-s-drivatar-actually-works
(8) Behaviour cloning issues pictured:
https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c
(9) Screenshot of the game Killer Instict
(10) Killer Instinct in game character art
(11) Melty blood type lumina character art:
https://typemoon.fandom.com/wiki/Kouma_Kishima/Remake
(12) Samurai shodown character art
(13) A sunset illuminating a dark cyberpunk city brightly in many colorful lights
generated by Dall-E 2