# SIGGRAPH'21 Course Notes
# New Techniques in Interactive Character Animation

Joan Llobera
joan.llobera@artanim.ch
Artanim Foundation
Geneva, Switzerland

Joe Booth
joe@joebooth.com
Independent Researcher
Seattle , USA

Caecilia Charbonnier
caecilia.charbonnier@artanim.ch
Artanim Foundation
Geneva, Switzerland

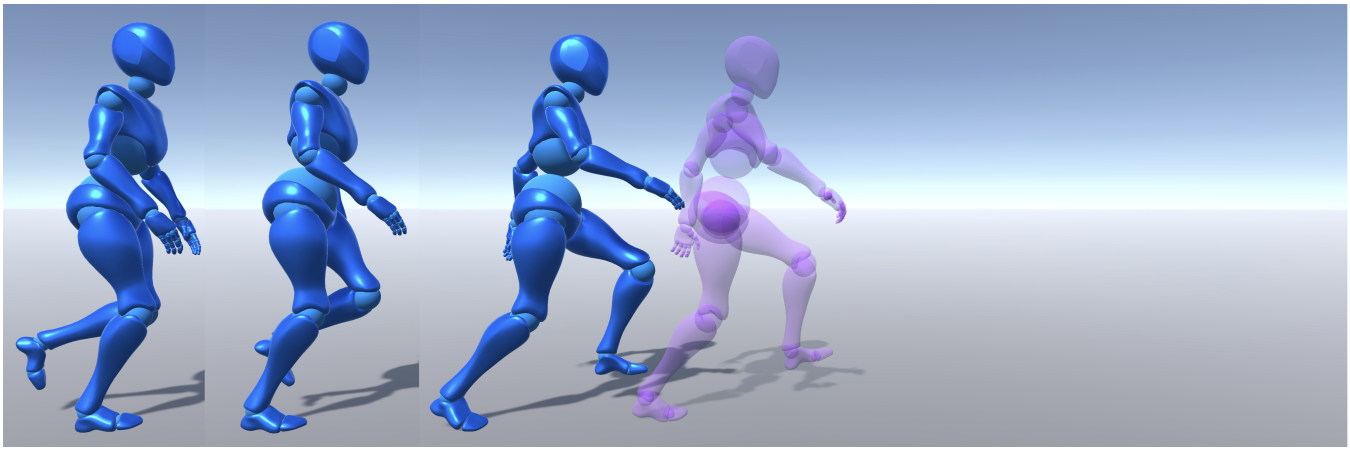

**Figure 1: A character animated through physical torques applied over a ragdoll imitating a reference cynematic animation**

## ABSTRACT

The application of deep learning for physics-based character animation and for cinematic controllers for interactive animation is changing how we should think about interactive character animation in video games and virtual reality. We will review the benefits and drawbacks of the techniques used and the implementations available to get started.

## CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

## KEYWORDS

Animation, Real-Time, Rendering, Machine Learning, Virtual Reality

**ACM Reference Format:**
Joan Llobera, Joe Booth, and Caecilia Charbonnier. 2021. SIGGRAPH'21 Course Notes New Techniques in Interactive Character Animation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Courses (SIGGRAPH '21 Courses), August 09-13, 2021.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3450508.3464604


*SIGGRAPH '21 Courses, August 09-13, 2021, Virtual Event, USA*

ACM ISBN 978-1-4503-8361-5/21/08.
https://doi.org/10.1145/3450508.3464604

## 1 COURSE OVERVIEW

Recent developments in machine learning are changing how we should think about interactive character animation in video game and virtual reality content production. In this talk we will review existing techniques, mainly focusing on two distinct but converging topics: physics-based character animation, and animation controllers for interactive animation. We will provide an overview of the main techniques that exist, what are the benefits and drawbacks for creating interactive animated characters, and what implementations of these techniques are available online.

Traditionally, character animation was based on blending cinematic trajectories created by animators. Transitions between animations were adjusted by hand, and interaction with the video game simulation was hand-crafted. Creating sophisticated interaction or dealing with many different animations often provoked different sorts of integration bottlenecks. Physics-based animation offers the potential to overcome all these limitations because the integration of physics-based characters within the video game dynamics is trivial.

Emerging animation controllers also simplify significantly the creation of interactive animation. Nowadays, the use of controllers trained on rich animation data sets allows to establish mappings ever more abstract between user input and constraints for animation synthesis. As a result, it is now possible to control sophisticated interactive characters using only high-level continuous input.

In this course we will review the main features of the solutions that have been proposed, on what technical insights they are based,

the benefits and drawbacks of each of them, and the main challenges remaining.

We will also review the main implementations available for artists and software engineers to get started. In particular, we will use the Marathon Environments project,[1] which contains some re-implementations of machine learning algorithms for interactive character animation, to illustrate some of the challenges found in different solutions proposed in the literature. The tool is available open source, with a liberal license, and we invite anyone interested in exploring their own solutions to use deep reinforcement learning for interactive character animation.

This course is designed for software engineers who want to start exploring how these techniques work, as well as for creative minds who want to use these techniques without having to dig too deep in the technical details.

## 2 COURSE CONTENT

### 1.Why physics-based characters?

*1.1. The challenge.* Implementing animation controllers introduces significant overhead in video game and virtual reality (VR) production. In production environments animation controllers can provoke many different kinds of bottlenecks. Moreover, the challenges they introduce are often found at the intersection of artistic skills (creating animations) and technical skills (programming the connection between user input and changes in the behaviour of the character).

Robust, more efficient methods for crafting interactive characters are actively sought to meet the ever-growing demand of video-game consumers. In addition, the arrival of of Immersive VR headsets to the consumer market introduces additional requirements to interactive character animation: VR users have a natural tendency to expect the interactive behaviour of virtual characters to be closer to natural social interaction rather than to video games.

In this course we will first outline the main methods by which interactive characters are typically crafted, and then review a series of innovations that have happened in the last 5 years, focused mainly on deep learning techniques applied to physics-based character animation and to innovative cinematic controllers. We will review the main contributions of different papers, the results they achieve, and what tools are available to the software engineer or the creative mind wishing to get started.

*1.2. Dominant Strategies in Industry .* Currently, the main method used to create interactive animations is based on segmenting animation sequences (see, for example, [1, 9]) , and defining points of transitions among them, which are generally maped to transitions in a state machine, or a hierarchical state machine. In this way, animation cycles or one-shot animations are triggered through changes in the state machine, which are themselves activated from logical conditions, determined by the game logic or by the user input.

These animation engines are generally available as built-in modules in what are the *de facto* standards of video game production (Unity3D and Unreal Engine). A relatively recent alternative is the use of *Motion Matching*, which we will review later on, as well as alternatives recently proposed for them. Some rare games (for example, *Totally Accurate Battle Simulator*, by Landfall Games), use physics-based animation, but those remain the exception, rather than the norm. In this course, while introducing the different techniques for interactive character animation, we will review to what extent they can be used to alleviate the challenges of creating interactive characters in video game production.

*1.3. Virtual Reality Challenges.* In VR, user input is not -or needs not to be- provided through game pads, mouses or keyboards. People immersed in VR move around, turn their heads, move their body, and they typically have wands that detect where their hands are, in addition to buttons. A head tracker detects in real time the head position and orientation. Moreover, despite rarely used in gaming, headsets provide microphones to introduce verbal input.

This multi modal, continuous input, supposes a considerable change compared to traditional game input. In addition, virtual characters appear much bigger, and much closer, to the user who, because of being immersed in a 3D simulation of the size of the real world, often has much higher expectations regarding the plausibility of the experience being rendered, something which becomes particularly challenging for interactive characters.

### 2.Try it out

We begin the course with a hands-on approach. outlining the basic steps to train a physics-based character, and the steps needed to create a new one based on an existing cinematic character. This part will be of most interest for artists interested in getting a first hand idea of how the methods discussed work in practice, and how physics-based character animation can fit within their process for integrating interactive animated characters.

### 3. Kinematic controllers

*3.1 Animation controllers without physics.* State Machines are still, today, the main method used to create animation controllers. Despite being robust, they present significant scaling problems, and are difficult to debug when a character involves many different, but related animations.

As a possible alternative, in 2013, Shoulson [15] showed how it was possible to combine different animation controllers in a way that allowed for more flexible blending of animations between tasks. This was made possible by using *choreographers*, which were like shadow poses which would synthesize the movements associated with different tasks. The important aspect to highlight is that the different poses would be combined according to a weight assigned to each task . In addition, a physics choreographer would adjust the resulting pose in order for it to be plausible physically, particularly regarding balance.

The system had problems in scaling, since combining different choreographers required some ad hoc coding. However, it already demonstrated that we can separate, on one hand, the idea of movement synthesis for a given task or a specific animation targeted and, on the other hand, the problem of making the movement compatible with physics. This direction has also been developed in articles already described earlier [2, 18].

[1] github.com/joanllobera/marathon-envs

In this section we will assume compliance with the physics of the environment is done in a second step of the character animation process, or that -as happens in certain scenarios- that it can be entirely avoided. We therefore focus specifically on the techniques (and demonstration projects) that are available to develop innovative animation controllers.

*3.2. Motion Matching (and improvements).* Motion Matching was proposed in 2015 by video game industry researchers [4] as a way to get smoother transitions between animations. At the time it appeared as a major innovation in how we could move away from state machines. It was based on calculating the distance between different poses of different animations. The ones with smaller distances were considered possible transitions. Once this (large) number of possible transitions was calculated, every frame or few frames the animation engine would look into which one among the possible transitions resulted in a pose closer to the targeted one, according to simple criteria, like the direction and orientation of the character if a transition to that animation chunk was performed. The system was shown robust, particularly for navigation tasks, and it has been adopted in an increasing number of commercial games. Different improvements have been proposed for motion matching, maybe most notable the recent contribution by Holden et al, [6], where the use of a neural network trained with a motion matching controller allows the same control with high-level controllers like a game pad, but without the need to load in memory all the motion data and possible transitions.

*3.3. Different neural architectures for kinematic animation controllers.* Deep learning has had a significant impact in how physics-based animation is thought. However, when we turn to animation controllers we find different supervised learning architectures, such as for example convolutional networks, LSTM architectures and, more recently, transformer-based architectures. What are the benefits and drawbacks of these? We review the most promising strategies, and the implementations available.

*3.4. Phase-based networks.* A neural architecture that is specific to animation controllers is the use of phase-functioned networks [7]. In phase-based networks, the training is associated with a cyclic input which is used to control the phase of the animation to be synthesised. For example,in a walking animation, the gait naturally lends itself for the definition of a phase. Training a network like this allows, at the synthesis step, to control quite well what phase of the movement the animation is, it therefore helps synthesising smooth transitions. However, this also limits the scope to which such methods can be applied, since it is limited to motions for which a global cycle can be determined. To put an extreme example, if we want to synthesise an animation of a character that goes forward on a mono-cycle while juggling, it would be impossible to define a global phase for the two behaviours. More dramatically, when cyclic behaviours are combined with non-cyclic behaviours (for example, walking while gesticulating in a conversation), such an approach cannot apply.

*3.5. Constraints, Contacts and Boundaries.* Traditionally, in inverse kinematics, to prevent the limbs from adopting impossible configurations, angular or positional constraints were introduced in the solving algorithm. A similar idea has been recently developed in [17] to solve the problem of global phases. By using the contacts in a motion capture data set, he manages to determine a *local* phase. In this way, a deep network can learn from motion capture data, and the intervals between contacts can be used to control the animation synthesised, but only for the joints affected. The authors demonstrate the idea very convincingly with motion data from basketball players, where the walking and bouncing movements can be controlled separately. Despite insofar this approach has not been tested on different motion data sets, it offers a quite promising approach.

*3.1 Background: 3D Rotations and Inverse Kinematics.* 3D Character animation is, almost universally, based on skeletal animation (At the exception of faces, which we will not cover here). As such, animations are based on combinations of 3D rotations. To develop efficiently physics-based character animation controllers, it is very useful to be at ease with how 3D rotations are implemented in a modern game engine, as well as the basics of Inverse Kinematics.

In modern game engines, 3D rotations are based on Quaternions, which generalise complex numbers. The benefit is that rotations can be represented in a very compact way (4 numbers), instead of with rotation matrices (12 components), and that they have interesting algebraic properties that make them suitable for simpler calculations. A drawback is that, just like complex numbers, they inhabit an abstract space with a circular topology, and it is therefore difficult to derive statistical measures meaningfully. For example, finding the average rotation of samples of a joint captured through time cannot be done with a simple weighted average, and more subtle methods must be found to average in the space of quaternion (see, for example,[10]) or otherwise make statistical calculations on spherical spaces (see, for example, [8]).

A detailed understanding of quaternion algebra is not needed to understand the main ideas of this course. However, to understand the technical detail of the implementations behind these solutions, it is necessary to be comfortable with these operations. We also recommend the person interested in such field to also get acquainted with the twist-swing decomposition, at least for the cases where decomposition is aligned with one of the main axis. A possible way to learn those systematically is going through the course materials and exercises that can be found here.

Inverse Kinematics (IK) was proposed as a solution to challenges such as how to find the rotations of different arm joints in order that the hand of a virtual character matches a given position. Initially, the methods proposed where based on calculation of Jacobian matrices or approaches based on gradient descent in multi-dimensional spaces, but ultimately heuristic methods proved more robust and intuitive, and they are most often used. Nowadays, the two main methods used are Cyclic Coordinate Descent and FABRIK, for which a summary description can be found in the slides supporting this course. We refer to the tutorial found in [3] for a general introduction to inverse kinematics, and to the FABRIK website[2], supported by it's main author, to know everything needed to use this IK method, including references and implementations. We would

[2]http://andreasaristidou.com/FABRIK.html

also like to point that IK is not a closed research area, with recent contributions, such as [16].

Despite widely used in industry, the practice of configuring how IK algorithms are applied on a given skeleton is tedious and, sometimes, cumbersome. Blending IK algorithms with animation sequences is always challenging: video game developers often must create heuristics, depending on the animation sequence being rendered, to apply IK solutions without deforming too much the animation sequence with which they are blended. It is therefore desirable to find a way to circumvent IK strategies, or to improve upon existing solutions to simplify the integration of these technique with the physics of the environment and the animation sequences rendered.

We provide links to resources for the person interested in getting a better understanding of these notions.

## 4. Create your own physics-based animation controller

*4.1. Rag dolls vs Rigged Characters.* Traditionally, in videogames rag dolls have been used when characters die, or fall. In those cases, we do not want the character to move following a predefined animation or trajectory. Therefore, to create an interactive character we used kinematic trajectories and, when some of these events happened, we activated the ragdoll behind it.

This has changed, recently, mostly because of the tremendous progress and accessibility of deep reinforcement learning techniques. These techniques have made possible to find which are the right forces and torques that need to be applied to ragdolls in order that the resulting animation matches a targeted animation. This change, suddenly, has open the way for combining traditional keyframe-based animation and production pipelines based on motion capture with physics-based characters.

*4.2.Reinforcement Learning meets physics-based animation.* Reinforcement Learning (RL) is a machine learning paradigm rooted in behavioural psychology, in which an agent performs actions, and perceives states of its environments. It learns on the basis of rewards, which are also provided by the environment.

Presented initially in 2013 [11], to then being published in top reference journals [12], agents showed model-free agents could learn super-human skills in classic atari videogames. Since then, the use of deep reinforcement learning has exploded, also unlocking a radical improvement in physics-based animation.

The initial benchmarks showed deep reinforcement learning could be used to have physical ragdolls walk, avoid obstacles, and progress in different ways. Those showed that a problem considered too difficult for traditional control theory could be addressed with deep reinforcement learning. In these systems, the agent would get positive rewards simply when the forces applied on the different limbs made the character move forward, and negatives rewards when it fell. However, the resulting characters were not usable in production environments, due to the fact that the style of movement was weird, and there was no obvious way to change it (for example, as shown in [5]).

This changed when Peng [13] showed that using a reference animation, and giving additional positive rewards also for having a pose similar to the reference animation, the agent could learn to move forward or not fall, while preserving the animation style of the movement produced. A particularly interesting aspect of this method is that it was simple: there were no particular assumptions on the topology of the joints, nor on the kind of movement, nor on the dynamics of the movement imitated. Target movements demonstrated including walking, running, but also acrobatic movements like air kicks. It worked on humanoids but also dinosaurs. The authors also showed that introducing a random parameter position at the training phase, with a positive reward for reaching it either with the body or with a limb, or even with a ball, allowed using the physics-based animation system as a controller: in the synthesis phase it was possible to impose the position of the target, and the movement was re-synthesised in a way that it reached the target.

Regarding limitations, the main was that the training had to be done on one specific animation, which meant that combining several animations required training a RL algorithm separately. At the inference time, all the different resulting physics controllers had to be loaded, and the system had to switch dynamically between one or another. It was also necessary to place the physics actuators manually, and the authors reported that doing so required some skill and intuition, it could not done automatically.

Since then a large variety of improvements has been introduced. For example, by introducing further flexibility in the input that the system can ingest [2], or providing a universal controller of the physical layer, where novel animations can be imitated, without the need to re-train the agent controlling the physical forces applied [18].

Overall, there are two main strategies to create physics-based character animation controllers, and those depend on whether at inference time there is a kinematic controller that the physics-based animation controller imitates, or if at inference time the entire animation is synthesised in a physics-based controller, without reference to a kinematic controller or to a reference animation. In

*4.3 Designing a reward.* What are the essential considerations that need to be you into account when designing a reward for a physics-controller based on RL? We here review possible strategies, depending on the kind of physics-based character animation controller you want to implement.

*4.4 Adjust the physical environment.* A considerable challenge, often overlooked, is how to adjust the physical environment where the RL agent learns. Should the agent learn to apply forces directly, or do so through a PD controller? Should we add noise in the training data? There is great variability in the existing literature regarding these aspects, and often they seem a bit overlooked in the contributions. However, these decisions have a major impact in the quality of the behaviour synthesised and they should not be overlooked.

*4.5. Adjust the training scenario.* Early termination, first introduced in [13], and recently further expanded in [18], has revealed to be a simple yet robust solution to the challenge of optimisation when combining multiple rewards. In general, when optimising parameters under different constraints, it is quite possible that solutions are not found because solutions that might be better when evaluated with the metric associated to one constraint can be worse for a metric associated to a different constraint. For example, applying certain forces at a given moment might result in a ragdoll imitating exactly a given cinematic trajectory, and therefore might give a

better reward in terms of pose similarity, but a worse reward in terms of physical stability. Therefore, the learning mechanism that might improve pose similarity could enter in conflict with the same learning mechanism applied on improving physical stability. Since we need the resulting system to do both, we need to find a way that the learning mechanism improves all the metrics *at the same time*.

The problem of optimisation under multiple constraints is, in general, difficult to address. However, here we can take advantage of the fact that the environment is a dynamic one, and simple reset the trial when *any* of the metrics is above a given threshold. In this way, the only solutions explored are the ones that satisfy all the constraints at the same time, with no possibility of conflict between them. Moreover, it also saves time, since there is a considerable space of possible solutions that is never explored: as soon as one of the reward metrics goes above a threshold, the solution is discarded.

## 5. It's your turn

*5.1. Open challenges.* A significant challenge in developing animation and physics controllers is animation sampling. As we saw, [13] trained a different controller for each animation. Soon after that publication, the same team demonstrated that motion extracted from video could be used to train the same system [14].

More recently, in [18] it was showed that using a large sample of animations on a given humanoid allowed for the generation of a physics controller that could imitate *any* animation controller for that humanoid. In practice, tests done internally to try to reproduce this paper suggest that this strategy can work as far as the controller is *smooth* (i.e., motion matching is fine, but triggering animations with a state machine introduces jumps that the physics controller cannot imitate). A bigger challenge is that the sampling of the animation space has to be done in a supervised manner: it is necessary to choose which are the right animations that have to be fed to the system, otherwise the system could over-learn some behaviours (for example, walking vs. doing acrobatic jumps), if there is more of such data in the training data.

The development of machine learning methods that generalise across a large database of animations is till a considerable challenge

*5.2 Your phyiscs-based controller.* Physics-based animation solves the challenge of integrating the behaviour of an interactive character within the video game simulation. However, animation controllers based on deep learning architectures still offer more flexibility to synthesise interactive behaviour. Despite this advantage, kinematic animation controllers require models that are significantly more complicated to put in place, when compared to RL architectures, and they still need to be integrated with a physical layer, either manually or with a learnt physics controller.

What is preventing us from combining both approaches? For example, can we use some of the animation controllers described in combination with a trained deep reinforcement learning system that converts those into physical actions? Alternatively, can we create a physics-based controller that does not depend on a kinematic controller, but that consistently responds to user input and can learn to imitate an arbitrarily large animation set? Which is the best answer to these questions is still unclear, but all the tools needed to answer are freely available for you to find the best answer for your particular goals.

## 6. Questions

If you have specific questions on the course contents, or on how to develop your own animation controller, please drop an email to get in touch!

## 3 ABOUT THE INSTRUCTOR

Dr Llobera has done research at the intersection of virtual reality and cognitive sciences for almost two decades. Initially trained in electrical engineering, with masters in cognitive sciences and software engineering, he did his PhD at Mel Slater's Virtual Reality Lab, and a Postdoc at Olaf Blanke's Cognitive Neuroscience Lab. He has also taught computer graphics and techniques for character animation at the ENTI videogame school, associated to University of Barcelona, and been a co-founder of two companies. He currently works at the Artanim Foundation as a senior researcher, bringing novel techniques of character animation to virtual reality experiences.

## 4 ACKNOWLEDGMENTS

We would like to thank the suggestions and animations provided by Valérie Juillard, which made useful suggestions during the exploration on how to reproduce the benchmarks outlined in this course, and crafted some of the animations that are integrated in the *Marathon Environments* benchmarks.

## REFERENCES

[1] Okan Arikan and David A Forsyth. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 483–490.
[2] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics (TOG)* 38, 6 (2019), 1–11.
[3] Ronan Boulic and Richard Kulpa. 2007. Inverse Kinematics and Kinetics for Virtual Humanoids.. In *Eurographics (Tutorials)*. 173–243.
[4] Simon Clavet. 2015. Motion Matching - The Road to NextGen Animation. *Proc of GDC* (2015). https://doi.org/watch?v=z_wpgHFSWss&t=658s
[5] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).
[6] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned motion matching. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 53–1.
[7] Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.
[8] John T Kent. 1982. The Fisher-Bingham distribution on the sphere. *Journal of the Royal Statistical Society: Series B (Methodological)* 44, 1 (1982), 71–80.
[9] Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 491–500.
[10] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. 2007. Averaging quaternions. *Journal of Guidance, Control, and Dynamics* 30, 4 (2007), 1193–1197.
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *Neural Information Processing* (2013). arXiv:1312.5602 [cs.LG]
[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, , Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236
[13] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

[14] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. 2018. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions On Graphics (TOG)* 37, 6 (2018), 1–14.
[15] Alexander Shoulson, Nathan Marshak, Mubbasir Kapadia, and Norman I Badler. 2013. Adapt: the agent developmentand prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics* 20, 7 (2013), 1035–1047.
[16] Sebastian Starke, Norman Hendrich, and Jianwei Zhang. 2018. Memetic evolution for generic full-body inverse kinematics in robotics and animation. *IEEE Transactions on Evolutionary Computation* 23, 3 (2018), 406–420.
[17] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 54–1.
[18] Tingwu Wang, Yunrong Guo, Maria Shugrina, and Sanja Fidler. 2020. UniCon: Universal Neural Controller For Physics-based Character Motion. arXiv:2011.15119 [cs.GR]

# A ONLINE RESOURCES TO GET STARTED

## A.1 Physics-based learning

Several work discussed in this course is available online open source, mostly with a non-commercial license. This allows the interested part to explore how the techniques described in the literature are actually used. An example of this is Deep Mimic, which shows the methods used for Peng's 2018 SIGGRAPH Article [13].

However, this kind of results are difficult to translate to a production environment, mainly due to the fact that they are not in one of the main game engines used in video game production. A possible solution for this challenge is the *Marathon Environments*[3] project, which allows anyone interested in physics-based learning to try different algorithms for deep reinforcement learning using an accessible game engine like Unity3D.

## A.2 Innovative Controllers

Starke has made available open source and free for research purpose some of the controllers discussed in this course [4]

We have also successfully combined *Marathon Environments* with a proprietary implemnentation of Motion Matching that is available at a modest price[5].

[3] https://github.com/joanllobera/marathon-envs/
[4] https://github.com/sebastianstarke/AI4Animation
[5] https://assetstore.unity.com/packages/tools/animation/motion-matching-for-unity-145624