# Boneshaker – A Generic Framework for Building Physical Therapy Games

**Lieven Van Audenaeren**
e-Media Lab, Groep T Leuven
Andreas Vesaliusstraat 13,
3000 Leuven, Belgium
Lieven.VdA@groept.be

**Jelle Husson**
e-Media Lab, Group T Leuven
Andreas Vesaliusstraat 13,
3000 Leuven, Belgium
Jelle.Husson@groept.be

**Vero Vanden Abeele**
e-Media Lab, Groep T/CUO
Andreas Vesaliusstraat 13,
3000 Leuven, Belgium
Vero.Vanden.Abeele@groept.be

**Jan-Henk Annema**
IBBT/CUO, KULeuven
Parkstraat 45 bus 3605
3000, Leuven, Belgium
JanHenk.Annema@soc.kuleuven.be

**Luc Geurts**
e-Media Lab, Group T Leuven
Andreas Vesaliusstraat 13,
3000 Leuven, Belgium
Luc.Geurts@groept.be

**Stef Desmet**
e-Media Lab, Group T Leuven
Andreas Vesaliusstraat 13,
3000 Leuven, Belgium
Stef.Desmet@groept.be


*CHI'12*, May 5–10, 2012, Austin, Texas, USA.
ACM 978-1-4503-1016-1/12/05.

## Abstract
We present the Boneshaker framework, a generic framework developed to facilitate the design of physical therapy games with the Unity 3D engine. The Boneshaker framework lowers the threshold for developing a variety of physical therapy games as it allows both developer and therapist to quickly add input devices and change specific game dynamics/therapy exercises.

## Keywords
Physical therapy games, Motion-sensing, 3D cameras

## ACM Classification Keywords
H.5.2 [User Interfaces]: Input devices and strategies

## General Terms
Design, Human Factors

## Introduction
In the past decade, we have seen a surge in sensors that can measure human movement. First the Nintendo® Wii-mote™, then the PlayStation® Move™ and finally the Microsoft® Kinect™ camera were released on the consumer market and rendered advanced motion-sensing systems available at a low cost. This also enabled academic research labs to tinker

with these technologies themselves [3,10,11]. At the same time, we have seen a turn to 'serious games' or those applications that aim to combine fun interaction with a serious purpose. The combination of 'gamification' and motion sensing has found a natural symbiosis in games for physical therapy. Commercially-of-the-shelf (COTS) sensors, open source software libraries, and development kits, have lowered the threshold for designing motion-based play. Indeed, during the past few years, several research projects have taken a similar endeavor of designing exertion games [8], addressing physical health and obesity [4], stroke rehabilitation [1], spasticity [9], cerebral palsy [6,9], etc.

Notwithstanding the mushrooming of academic realizations of physical therapy games, few of these have reached commercial success. Our own experience in building and commercializing physical therapy games [5] has unveiled two major obstacles that need to be overcome:

1) The inherent limitations of every COTS device, which affects the range of specific physical exercises that can be executed and measured with adequate quality and/or reliability. This opposes the typical nature of physical therapy where a therapist is looking for a variety of exercises, even with one patient or in one therapy session [2].

2) The short half-life of motion-sensing technology, which often renders a therapeutic game obsolete by the time it has reached the market.

## The Boneshaker framework

With the Boneshaker framework, we aim to build on the lessons learned from a previous research project on building physical therapy games [2,6,9] and to resolve the problems concerning the short half-life of motion-sensing technologies while increasing the variety of physical therapy exercises. The framework is not just a software development kit for one specific input device, but rather a layer on top of that. This layer, situated between the device drivers of motion-sensing technology and the game itself (see **figure 1**), gives both the game developer and the therapist advantages.


**figure 1.** The Boneshaker framework is a layer between input device and the game.

Firstly, it simplifies creating support for multiple input devices. The data coming from Microsoft® Kinect™, the Nintendo® Wii-mote™, or any other input device is mapped onto a *skeleton schema* (see **figure 2**). This *skeleton* has been defined in the eXtensible Markup Language (XML). Consequently, the game developer can easily change the skeleton structure itself, without having to create a new version of the game. For example, in the likely situation of new input devices that allow sensing at a higher resolution, the developer can define an enhanced skeleton. This new skeleton, containing more joints, is created by simply adapting the XML schema. The actual game logic within the game does not need to be adjusted.

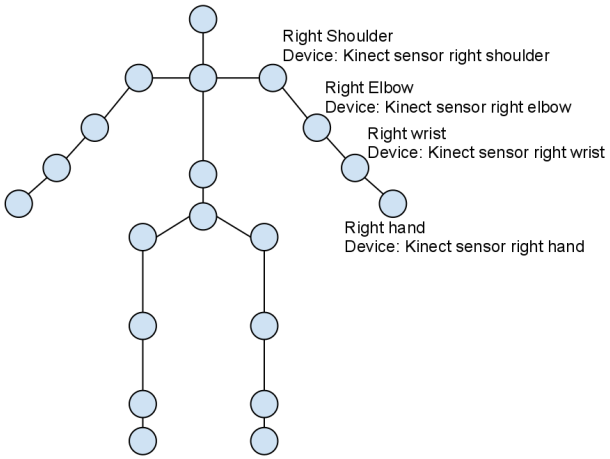


**figure 2**. Example of the skeleton interface, every joint that is selected has a name and a device assigned to.

Then, a game is connected to this skeleton by defining which type of input the game needs, e.g. the number and type of joints, and whether those joints make use of a relative or absolute position, an angle, a translation, an acceleration or a weighted sum of movements, etc. How the joints' information is translated to the type of input the game needs is defined in the *game dynamic* [7][1] *schema*.

The game dynamics are also defined in XML, resulting in a system that is very flexible. When a game dynamic needs to be changed, e.g. when a therapist wants the movement in the sagittal plane only, instead of the coronal plane, or when the upper limbs should be captured rather than the lower limbs, this is simply changed in the game dynamic schema.

Within the game itself, a visualization of this skeleton schema and the associated game dynamics schema can also be accessed by the therapist. He or she can specify which body parts/ joints/angles will be required for a specific exercise/game. Consequently, this allows the therapist to use one game for multiple physical exercises, by simply changing the chosen joints and adapting the wanted moves. Every game can be adjusted for every player and according to the specific needs of the patient.

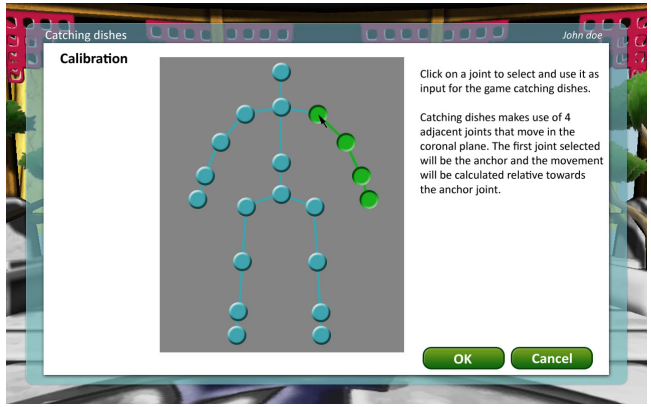


**figure 3**. A screenshot of the GUI for the therapist.

In sum, both the developer and the therapist can easily change the input for a certain game, try out a new control scheme or create a new therapeutic exercise to play the game.

[1] Since this game dynamic schema truly describes how the interaction of the player with the game is translated in in-game action, we call this the game dynamic, inspired by the MDA framework by Hunicke et al.[7].

## Architecture of the Boneshaker framework

The architecture used for the Boneshaker framework is shown in **figure 4**, which is a diagram that represents the internal structure of the Boneshaker framework. The five main building blocks of the Boneshaker framework are:

1. Driver, a specific input device driver for the Boneshaker framework.
2. Boneshaker device, for example the acceleration sensors from the Nintendo® Wii-mote™ or 3D coordinates from the Microsoft® Kinect™ camera.
3. Joint, the building blocks of the skeleton.
4. Skeleton, a collection of joints that represent the human body.
5. Game dynamic, a definition of which input is needed for and how it is used in a game.

These five building blocks are the core of the Boneshaker framework. Every block has its specific purpose, in the next paragraphs we will explain in detail why they were created and how they work.

*Driver*

A driver is necessary to get the needed input data from the input device that is used. Every input device has its own drivers, created by its developers. Unfortunately, not every input device is standard; moreover a lot of input devices provide data in different representations. Therefore the Boneshaker driver is used to translate the original device driver data to data that can be used by the framework. This is the positional or rotational data of a skeleton joint. The Boneshaker driver can be used with multiple devices, as the user should be able to use multiple identical input devices for different joints. For example, the user can use two Wii-motes™, one for the left and one for the right hand, however both devices need the same translation from device-data to Boneshaker-data.

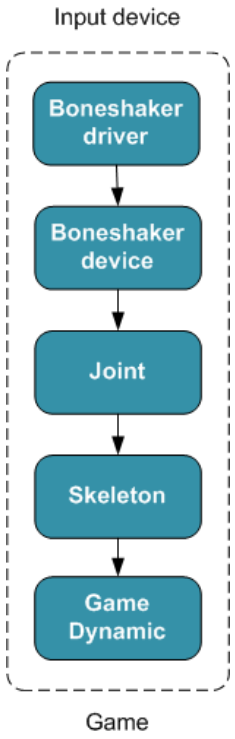


**figure 4.** Internal structure of the Boneshaker framework

*Device*
A Boneshaker driver, a name and an output type define a device. The output type specifies whether the device is delivering positional or rotational data. This definition is done in XML and can be changed on the fly.
During runtime the device will also poll the data from the Boneshaker driver to update the input data internally. This data is then used in the joints of the skeleton.

*Joint*
The joints are the building blocks of the skeleton, which represents the human body playing the game. Every joint is defined by a name, possibly a device attached to the specific joint and possibly a parent joint and/or children joints, as is shown in **figure 2**. For example, the right wrist joint has the right elbow joint as its parent and the right hand as its child. While the right hand joint has no children, its parent is the right elbow joint.
All this data is also defined in XML, so the device attached to a certain joint can be changed on the fly by the developer. During runtime a joint gets the selected device data from its assigned device and transforms it to the wanted representation.

*Skeleton*
The skeleton is a collection of joints that represent the player's human body.

*Game dynamic*
The game dynamic is a schema that holds information about how to translate the data from the collection of joints a in a skeleton to the type of input the game needs. The philosophy behind a Game dynamic is that it embodies the logic necessary to assess the physical gesture made by the player. The same Game dynamic can then be assigned to multiple games. A Game dynamic is also defined in XML and its definition contains a name, a list of joints needed and the name of the function that does the transformation from joint data to data the game can read.

## Discussion
While we have the aim to make the Boneshaker framework as universal as possible, at this moment, the framework has been tested with the Unity 3D game engine only. The choice for Unity is a result of it being the game engine for a previous research project on building physical therapy games [2,6,9]. While this may seem an opportunistic reason, we are convinced that Unity 3D is a good choice as an affordable yet flexible game engine that allows researchers and smaller organizations to develop physical therapy games.

Currently, Boneshaker has been tested to work with a 3D camera. The 3D camera of our choice was the Microsoft® Kinect™ because the device is well-known, it was reasonably priced and we had Microsoft's® official SDK to work with. Obviously, in the future, we aim to expand different input devices as well. A first effort will be to connect the framework with the Nintendo® Wii-mote™ and the Nintendo® Balance board™, as these input devices are present in many practices of physical therapists already, and provide sensor data that can be complementary to data coming from a 3D camera.

## Conclusion
The Boneshaker framework is the result of previous research project on developing physical therapy games. We unveiled a need to overcome the short half-life of

motion-based input devices as well as the need for a greater variety in game dynamics/physical exercises. Therefore, we developed a layer between the input device and the game to increase the flexibility of the game. Firstly, it facilitates extending the game to multiple input devices and/or adding the latest versions of input-devices by using an easily adaptable skeleton schema defined in XML. Secondly, it enables using multiple game dynamics with one input device/game by providing a "game dynamics" schema that holds information about how to translate the data from the collection of joints a in a skeleton to the type of input the game needs. Hence, by using the Boneshaker framework, both the developer and the therapist can change the input for a game, try out a new control scheme or create a new therapeutic exercise for a game.

## References


1. Alankus, G., Lazar, A., May, M., and Kelleher, C. Towards customizable games for stroke rehabilitation. *Proceedings of the 28th international conference on Human factors in computing systems*, ACM (2010), 2113-2122.
2. Annema, J.-H., Verstraete, M., Vanden Abeele, V., Desmet, S., and Geerts, D. Videogames in therapy: a therapist's perspective. *Proceedings of the 3rd International Conference on Fun and Games*, ACM (2010), 94–98.
3. Chung Lee, J. Projects - Wii. 2008. http://johnnylee.net/projects/wii/.
4. Deangelis, G. Combating Child Obesity: Helping Kids Feel Better by Doing What They Love. 2008. http://www.gamasutra.com/view/feature/3692/combating_child_obesity_helping_.php.
5. Geurts, L., Vanden Abeele, V., Husson, J., et al. Digital Games for Physical Therapy: Fulfilling the Need for Calibration and Adaptation. *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, ACM (2011).
6. Geurts, L., Vanden Abeele, V., Husson, J., et al. Digital Games for Physical Therapy: Fulfilling the Need for Calibration and Adaptation. *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, ACM (2011).
7. Hunicke, R., LeBlanc, M., and Zubek, R. MDA: A Formal Approach to Game Design and Game Research. (2001).
8. Mueller, F., Agamanolis, S., and Picard, R. Exertion interfaces: sports over a distance for social bonding and fun. *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (2003), 561-568.
9. Vanden Abeele, V., Geurts, L., Husson, J., et al. Designing Slow Fun! Physical Therapy Games to Remedy the Negative Consequences of Spasticity. *Proceedings of the 3rd International Conference on Fun and Games*, ACM Press (2010).
10. Wii homebrew. http://wiibrew.org.
11. OpenNI. http://www.openni.org/images/stories/pdf/OpenNI_UserGuide.pdf.