Multimedia Tools and Applications (2021) 80:27947–27971
https://doi.org/10.1007/s11042-021-10980-3

PhyDSL$_K$: a model-driven framework for generating exergames

Maria Teresa Baldassarre[1] · Danilo Caivano[1] · Simone Romano[1] · Francesco Cagnetta[1] · Victor Fernandez-Cervantes[2] · Eleni Stroulia[2]

Received: 1 September 2020 / Revised: 31 December 2020 / Accepted: 30 April 2021 /
Published online: 27 May 2021

Abstract

In recent years, we have been witnessing a rapid increase of research on exergames—*i.e.,* computer games that require users to move during gameplay as a form of physical activity and rehabilitation. Properly balancing the need to develop an effective exercise activity with the requirements for a smooth interaction with the software system and an engaging game experience is a challenge. Model-driven software engineering enables the fast prototyping of multiple system variants, which can be very useful for exergame development. In this paper, we propose a framework, *PhyDSL$_K$*, which eases the development process of personalized and engaging Kinect-based exergames for rehabilitation purposes, providing high-level tools that abstract the technical details of using the Kinect sensor and allows developers to focus on the game design and user experience. The system relies on model-driven software engineering technologies and is made of two main components: *(i)* an authoring environment relying on a domain-specific language to define the exergame model encapsulating the gameplay that the exergame designer has envisioned and *(ii)* a code generator that transforms the exergame model into executable code. To validate our approach, we performed a preliminary empirical evaluation addressing development effort and usability of the PhyDSL$_K$ framework. The results are promising and provide evidence that people with no experience in game development are able to create exergames with different complexity levels in one hour, after a less-than-two-hour training on PhyDSL$_K$. Also, they consider PhyDSL$_K$ usable regardless of the exergame complexity.

Keywords Model-driven game development · Rehabilitation exergames · Model-driven software engineering

✉ Maria Teresa Baldassarre
mariateresa.baldassarre@uniba.it

[1] University of Bari, Bari, Italy

[2] Department of Computing Science, University of Alberta, Edmonton, Canada

1 Introduction

Exergames are video-games that require body movements to play and are thought as a form of physical activity [12]. In recent years, research in the context of exergames has mainly focused on rehabilitation purposes, although rehabilitation exergames are often not engaging or challenging enough, resulting in a failing therapy. It is our opinion that to overcome these difficulties, the development process of exergames should be user-centered and not be effort prone nor require specific programming skills.

To develop games (and thus exergames), developers have to typically translate their mental models of the envisioned gameplay into code, which is a challenging task since General-Purpose Languages (GLPs), like Java or C, are not designed to capture such models [13]. On the other hand, Domain-Specific Languages (DSLs) can be tailored to provide syntactic constructs that correspond to the requirements that match developers' needs for a specific application domain. DSLs are used in Model-Driven Software Engineering (MDSE) technologies to define models that, after applying transformation chains, are translated into artifacts (*e.g.,* code for a specific platform). MDSE technologies can be exploited to ease the game development process (and thus the exergames development process), bridging the gap between the semantics of GLPs and concepts of a specific application domain. Bearing this in mind, Guana et al. [13] designed and then implemented a DSL, *PhyDSL*, to allows developers to naturally define gameplay models of 2D physics-based games, reflecting the mental models of the developers. They also implemented a transformation chain to (automatically) translate the defined models into executable code for Android[1] devices.

In this paper, we describe our experience in developing a new transformation chain for a second target platform of *PhyDSL* [13]. Keeping in mind that PhyDSL was conceived especially for developers with little programming experience [13] and motivated by the need to ease the development of exergames for upper-limb rehabilitation exercises, we developed a second transformation chain, in $PhyDSL_K$, to enable the generation of exergames for the Kinect sensor and the Unity game engine with Windows as the target platform. The PhyDSL_K framework is made of two main components: *(i)* an authoring environment relying on a DSL, PhyDSL, to define the exergame model encapsulating the gameplay that the exergame designer has envisioned and *(ii)* a code generator that transforms the exergame model into executable code. The complete natural interface provided by Kinect-based exergames can obtain better results as compared to mixed interfaces where users have to interact with their body along with physical devices such as game controllers or balance boards. This is especially true for seniors—the end-users of our exergames—, who are usually not familiar with these devices and may be reluctant to learn how to use them. Moreover, the usefulness of a DSL, like PhyDSL, should increase if it is supported by multiple transformation chains leading to code generation for multiple platforms (like in our case).

To validate our approach, we performed a preliminary empirical evaluation addressing development effort and usability of PhyDSL_K. To that end, we asked 14 participants with no experience in game development (and thus exergame development) to develop three exergames with increasing complexity by using PhyDSL_K.

The key contributions of this paper can be summarized as follow:

– A new transformation chain to enable the generation of exergames for the Unity game engine with Windows as the target platform and user-interaction modality based on Kinect.

[1] https://www.android.com

– Three exergames for upper-limb therapies for seniors, representative of the types of exergames that the PhyDSL$_K$ system can produce. Each game has different functionality and different complexity, enabling different types of movements to be accomplished by the users to reach the goals of the therapies.
– A preliminary empirical study that evaluates PhyDSL$_K$ with respect to development effort and usability. The validation aimed to investigate whether it is feasible for people with no experience in game development to create exergames with PhyDSL$_K$ regardless of the complexity of the exergame.

The remainder of this paper is structured as follows. Section 2 provides the main concepts on MDSE and summarizes work related to ours. In Sections 3 and 4, we introduce PhyDSL$_K$ and three exergames developed by using PhyDSL$_K$, respectively. The preliminary empirical assessment of PhyDSL$_K$ and the obtained results are presented in Section 5. Final remarks conclude the paper, along with our future research agenda.

2 Background and related work

In this section, we review the key concepts on MDSE and discuss an example of application of MDSE, namely PhyDSL, the original DSL and code-generation engine on which our work is based. We then summarize work on the use of exergames in the fields of physical medicine and rehabilitation, along with work aiming to improve the development of exergames.

2.1 Model-driven software engineering and PhyDSL

MDSE practices can increase efficiency and effectiveness in software development, creating a direct line with human mind computational tasks such as abstraction, or, in the more MDSE specific term, modeling [4]. The core concepts in MDSE are models and transformations, which need to be expressed in an appropriate modeling language. Modeling languages are defined using three main components: *(i)* abstract syntax, which describes the language and its structure as well as how primitives can be combined at the highest level of abstraction without taking into account any representation or encoding; *(ii)* concrete syntax, which describes the exact representation of the language; *(iii)* and semantics, which describes the sense of any element of the language and any combination of them.

In Fig. 1, we show an overview of the MDSE methodology, as a top-down process. Models are defined according to a DSL (*i.e.,* a programming language built for a specific domain), in turn, defined using a meta-modeling language. Then transformations are executed according to transformation rules specified through a transformation language. In the end, the artifacts produced can be deployed to the desired platform.

Guana et al. [13] exploited MDSE to ease the development of 2D physics-based games for Android devices. The core of this work is PhyDSL, a DSL that enables game designers to define gameplay models of 2D physics-based games, reflecting the mental models of the designers. Guana et al. also implemented a transformation chain that takes as input the defined gameplay models and produces executable code for Android devices. The PhyDSL language and code-generation engine rely on a point-and-click user-interaction model, appropriate for the Android platform; instead, PhyDSL$_K$ adopts the Kinect sensor as the input device, responding to the user's movements to control the game, whose state is reflected on a Unity-based environment.

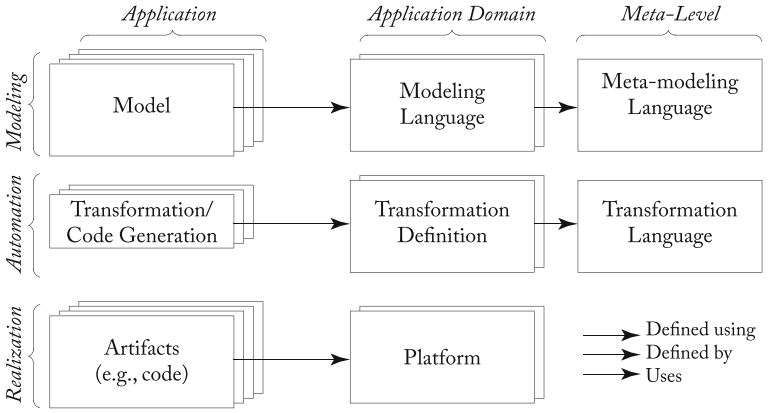

Fig. 1 Overview of the MDSE methodology (top-down process) [4]

2.2 Exergames

In recent years, exergames have seen ever-increasing popularity in the video game market. Nintendo Wii and Microsoft Xbox 360 consoles have improved the gameplay paradigm by introducing new interactive control systems [9]. While the Wii console allows users to interact with games via a remote controller, the Xbox 360 console provides a different and more immersive experience thanks to the Kinect sensor, which allows users to control the game solely through their bodies—*i.e.,* without remote controllers. In 2012, Microsoft released Kinect for Windows along with the Kinect SDK so allowing researchers to explore applications of the Kinect sensor in fields different from gaming. For example, researchers have studied applications of the Kinect sensor in the fields of physical medicine and rehabilitation (*e.g.,* [15, 19]) since Kinect-based exergames allow setting up a cost-effective and entertaining exercise system able to collect quantitative data about users' movements, calorie consumption, and aerobic activity [22]. Lange et al. [15] observed that people with limited experience in video games were excited to play a Kinect-based rehabilitation exergame within a clinical setting. Pastor et al. [19] reported that a stroke survivor was engaged with a Kinect-based exergame designed for upper-limb rehabilitation and that she was willing to use it at home. Sáenz-de-Urturi et al. [21] showed how the participation of domain's experts at the design phase of a Kinect-based exergame can lead to engaging and effective gameplay for elder people. Galna et al. [11] assessed the accuracy of the Kinect sensor in measuring movements in people with Parkinson's disease and showed that Kinect has the potential to become a home-based, cost-effective sensor to measure movements in people with Parkinson's disease, despite it is not so accurate in measuring small movements such as toe-tapping. Averell and Know [1] proposed a rhythm-based music game technology to support stroke rehabilitation and highlighted how the proposed technology was capable of monitoring, thanks to the Kinect sensor, the progress of stroke survivors. Ofli et al. [18] showed that Kinect was a viable means to monitor elder people while doing gym exercise at home. Li et al. [16] proposed a system that included rehabilitation exergames for elder people and reported that the participants involved in a preliminary study were favorable to use the system in the future. Despite the above-mentioned studies [1, 11, 15, 16, 18, 19, 21] investigate applications of the Kinect sensor in the fields of physical medicine and

rehabilitation (just like ours) with encouraging results, none of them presents supporting tools for developing rehabilitation exergames (unlike our study).

The results of the above-mentioned studies [1, 11, 15, 16, 18, 19, 21] suggest that the Kinect sensor can be successfully applied in the fields of physical medicine and rehabilitation; however, there are still some obstacles in developing engaging exergames efficiently. Fernandez-Cervantes et al. [8] developed *VirtualGym*, a system that aims to simplify the development process of Kinect-based rehabilitation exergames by providing a language for specifying postures and movements. The system includes an editor that enables domain experts to specify the rehabilitation exercise (*i.e.,* a gym exercise) by editing a virtual demonstration of that exercise. During the exergame, a virtual coach avatar demonstrates the exercise so that the patient can learn and practice the exercise. The movements of the patient are then reflected in real-time on the avatar. Feedback is provided to the patient about the correctness of the execution of the exercise. Fernandez-Cervantes et al. also conducted a preliminary study with potential patients who were asked to do exercises by using VirtualGym; the study results enabled an improvement of the system interface. While VirtualGym supports the development of rehabilitation exergames in the form of gym exercises, PhyDSL$_K$ allows developing 2D physics-based exergames (*e.g.,* casual games) for rehabilitation purposes. Also, we evaluated our system by focusing on the development of exergames, unlike Fernandez-Cervantes et al. who evaluated their system by focusing on how it supports patients in doing rehabilitation exercises.

Portes et al. [20] proposed a system to ease the development of rehabilitation exergames for children/teenagers suffering from lower back pain. This system is based on a high-level language, named *PEL*, and allows developing exergames in the form of gym exercise. The therapist defines the exercise requirements (*e.g.,* the trajectory of the rehabilitation movement), which a developer translates into PEL for then automatically generating the exergame. The system also aids the patient in doing the exercise. In particular, a virtual coach avatar shows the exercise that the patient had to then repeat. During the exercise execution, the system monitors the performance of the patient through the Kinect sensor. Portes et al. also conducted a preliminary study with potential patients who were asked to do an exercise by using the system. The results suggest that the system facilitates the rehabilitation process of patients. One of the main differences between Portes et al. 's system and PhyDSL$_K$ is the kind of supported exergames (*i.e.,* gym exercises in the form of exergames vs. 2D physics-based exergames for rehabilitation purposes). Also, unlike Portes et al., we assessed our system with respect to development effort and usability.

Mocanu et al. [17] proposed a system to stimulate physical activity adapted to elder people. The system is based on Kinect and allows creating an exergame where the virtual patient avatar had to reproduce the gym exercise showed by the virtual coach avatar—*i.e.,* gym exercises in the form of exergames. The system allows the therapist to record a gym exercise that the coach avatar will reproduce. The results from a preliminary evaluation of the system show that the system can engage patients in physical activity for a long time—Mocanu et al. did not evaluate development effort and usability of their system.

Hardy et al. [14] proposed a framework for the development and use of customized and adaptive exergames to train elderly and disabled people. The framework allows experts of different domains to be involved in the design phase together with game designers, exploiting an authoring environment. Hardy et al. showed the capabilities of the framework by developing two exergames; the former had an interaction modality based on an ergometer bike, the latter had an interaction modality based on a balance board. Afterward, the authors evaluated the developed games by involving potential patients and reported that the

participants accepted the games as an appropriate kind of training. Again, the authors did not evaluate their system with respect to development effort and usability. Also, Hardy et al. 's system allows developing exergames different from PhyDSL$_K$ (*e.g.,* the exergames developed through PhyDSL$_K$ are Kinect-based, unlike those developed through Hardy et al. 's system).

The above-mentioned studies [8, 14, 17, 20] aim to increase the direct contribution of domain experts in the design stage of exergames, trying to scale down the problem-implementation gap [10]. That said, we believe this is the right direction to undertake and more effort is needed to build a complete model-driven system able to address the overall complexity of the exergame development process, providing the appropriate abstraction levels that allow domain experts to be autonomous protagonists in all phases from design to deploy. Based on this perspective, we developed PhyDSL$_K$, which we describe in the next section.

3 PhyDSL$_K$

The development process of an exergame through PhyDSL$_K$ is depicted in Fig. 2 by using a UML activity diagram with object flow where rounded rectangles represent phases and rectangles represent objects consumed/produced by these phases. As depicted in Fig. 2, the development process of an exergame through PhyDSL$_K$ is a pipeline made up of two main phases:

1. **Model Specification**. This (main) phase allows the exergame designer to create the exergame model encapsulating the gameplay she has envisioned. The PhyDSL language [13] is used to specify the exergame model. This phase comprises eight sub-phases (see Fig. 2), each of which is conveniently recalled in the following of this section, and produces the exergame model as the output.
2. **Code Generation**. This (main) phase takes the exergame model and applies M2T transformations, which enable the automatic generation of code (*i.e.,* scripts) ready for the Unity game engine with Windows as the target platform for the exergame and user-interaction modality based on Kinect. The scripts generated by these M2T transformations are the output of this phase.

We detail these two phases in the rest of this section.

3.1 Model specification phase

To allow the exergame designer to specify the exergame model, we reused the PhyDSL plugin for the Eclipse IDE by Guana et al. [13]. The plugin requires the exergame designer to first create a gameplay design file (*i.e.,* a file with the *phy* extension)—the sub-phase **Creation of Gameplay Design File** in Fig. 2. After creating the gameplay design file, the exergame designer can specify the exergame model with the support of the plugin, which extends the Eclipse IDE by providing a text editor for the PhyDSL language including automatic-completion and static-checking of PhyDSL code (*i.e.,* code saved in phy files). In Fig. 3, we show these two features of the PhyDSL plugin.

Guana et al. conceived the PhyDSL language to help the game designer to translate her mental model of the gameplay into code [13]. To that end, the language requires the exergame designer to provide an answer to five questions, each of which is associated with a concept of the gameplay. The answers to these questions are provided through five gameplay

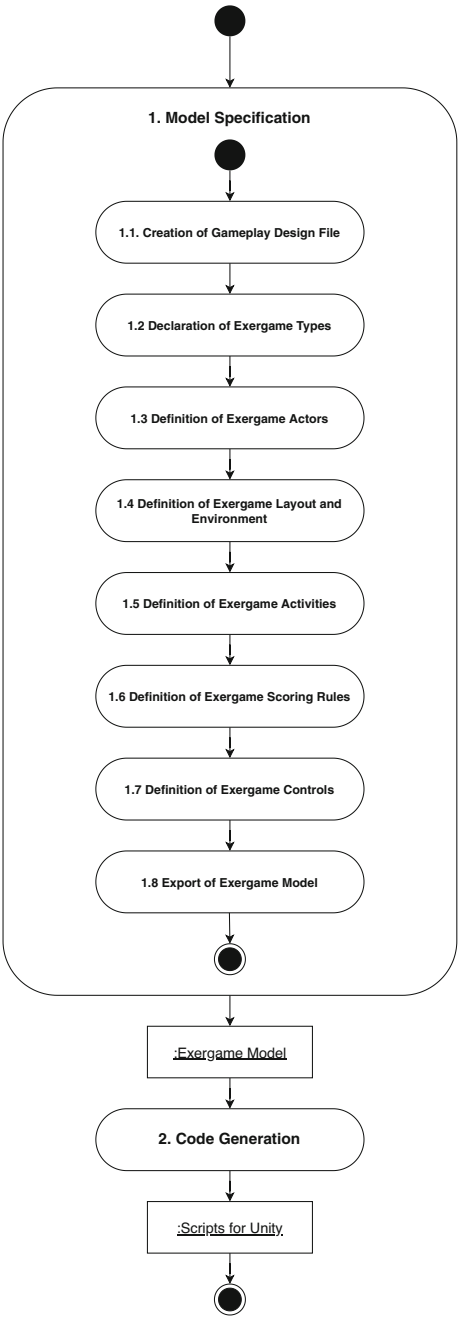

Fig. 2 Development process of an exergame through PhyDSL_K

definition sections, respectively. The questions and the corresponding gameplay definition sections are shown in Table 1.

The PhyDSL language is based on enumerated types to specify concrete values for the variables that will be then used to define gameplay concepts (*e.g.,* actors or layout). The

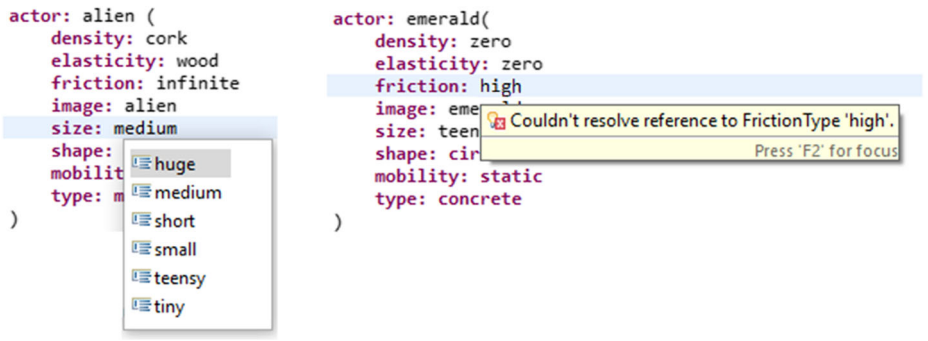

Fig. 3 Automatic-completion (on the left-hand side) and static-checking (on the right-hand side) of PhyDSL code in the PhyDSL plugin

Types definition section is where exergame designers specify these types—the sub-phase **Declaration of Exergame Types** in Fig. 2. In Fig. 4, we show an example of the Types definition section for an actual exergame, namely *AlienMiner*.

In the Actors definition section, it is possible to define the exergame actors—the sub-phase **Definition of Exergame Actors** in Fig. 2—along with their properties, which are: density, elasticity, friction, image, size, shape, mobility, and type. In Fig. 5, we show the Actors definition section for the AlienMiner exergame. The density, elasticity, and friction properties specify how the actor will interact with physics; while image and shape specify its appearance. The value of the shape property (*i.e.,* either circle or square) is also used to determine how to manage the collisions. The mobility property can be either dynamic or static. The former will let actors be affected by physical forces such as gravity and collisions, while the latter will not. Finally, the type property is used to specify if the actor will be abstract, concrete, or main. An abstract actor will not interact with other actors and its physical properties will be ignored by the game engine. A concrete actor will support all physical interactions. The main actor will be the one controlled by the player and, if a continuous camera is defined (see the Layout and Environment definition section), it will be followed by the camera.

In the Layout and Environment definition section, exergame designers can specify where to locate the actors of the exergame, as well as the gravity, background image(s), touch capabilities, and camera behaviors—the sub-phase **Definition of Exergame Layout and Environment** in Fig. 2. In Fig. 6, we provide an example of the Layout and Environment definition section for the AlienMiner exergame. The locations of the actors can be specified by using the coordinate system of Unity where 1 unit is equal to 100 pixels and the origin of

Table 1 Questions related to gameplay concepts with the corresponding gameplay definition sections

Questions related to gameplay concepts	Gameplay definition sections
Q1: Who is the player?	S1: Actors
Q2: Where does the player live?	S2: Layout and Environment
Q3: What are player's challenges?	S3: Activities
Q4: What are player's goals?	S4: Scoring Rules
Q5: What are player's available actions?	S5: Controls

```
Types:

 // Resources
 resource alien: "alien"
 resource meteorite: "asteroid_fire"
 resource emerald: "emerald"
 resource star: "level_end"
 resource diamond: "diamond"
 resource brick: "granite"
 resource planet: "alien_planet1"
 resource portal: "spatial_portal"
 resource aBtn: "a_button"
 resource shot: "shot"
 resource crash: "glass_koenig"

 // Elasticity
 elasticity zero: 0.0
 elasticity rock: 0.05
 elasticity wood: 0.5
 elasticity tennisBall: 2.0
 elasticity beachBall: 4.0

 //Friction (us)
 friction zero: 0.0
 friction teflon: 0.04
 friction ice: 0.1
 friction brick: 0.4
 friction rough: 0.7
 friction infinite: 100.0

 // Density g/cm^3
 density zero: 0.00
 density cork: 1.12
 density rubber: 1.2
 density stone: 2.3
 density granite: 2.6
 density glass: 2.7
 density lean: 11.3

 // Linear speed
 linear speed fast: (40.0, 40.0)
 linear speed slow: (10.0, 10.0)

 // Angular Velocity
 angular velocity fast: 40.0
 angular velocity slow: 10.0

 // Gravity
 gravity moon: 4.0
 gravity earth: 9.8

 // Actor Sizes
 size tiny: 0.2
 size teensy: 0.3
 size short: 0.4
 size medium: 0.45
 size small: 2.0
 size huge: 8.0

 // Actions
 force up: (0.0, -5.0)
 force right: (2.0, 0.0)
 force left: (-2.0, 0.0)
 force upRight: (2.0, 2.0)
 force upLeft: (-2.0, 2.0)
```

Fig. 4 An example of Types definition section for AlienMiner

the x and y axes is located in the middle of the first background image—each background image is resized to 1920x1080 pixels. As for the gravity, it is merely set by choosing one of the gravity values defined in the Types definition section. The background is set thanks to two parameters: image and segments. The former is related to the custom resource type defined in the Types definition section and refers to image file(s). The latter lets the game engine understand how many images will compose the background. The exergame designer, indeed, can use multiple images that will be horizontally tiled. Exergame designers must

```
Game AlienMiner:

 design actors:

   actor: alien (
     density: cork
     elasticity: wood
     friction: infinite
     image: alien
     size: medium
     shape: circle
     mobility: dynamic
     type: main actor )

   actor: meteorite (
     density: granite
     elasticity: wood
     friction: rough
     image: meteorite
     size: small
     shape: circle
     mobility: dynamic
     type: concrete )

   actor: brick (
     density: cork
     elasticity: rock
     friction: brick
     image:brick
     size: teensy
     shape: square
     mobility: static
     type: concrete )

   actor: emerald (
     density: zero
     elasticity: zero
     friction: zero
     image: emerald
     size: teensy
     shape: circle
     mobility: static
     type: concrete )
 ...
```

Fig. 5 An example of Actors definition section for AlienMiner

```
design layout:

  locate: alien (1.0, 1.0)
  locate: brick (5.0, 3.0)
  locate: brick (5.5, 3.0)
  locate: brick (6.0, 3.0)
  locate: brick (5.0, 6.5)
  locate: brick (5.5, 6.5)
  locate: brick (6.0, 6.5)
  locate: brick (6.5, 6.5)
...
design environment:
  gravity: moon
  background: (image: planet
       segments:9)
  touchscreen enabled: false
  camera: continuous

```

Fig. 6 Example of Layout and Environment definition section for AlienMiner

follow a naming convention to correctly build the background: consecutive numbers must be put at the end of the file names (*e.g.,* bg1, bg2, bg3, and so on). The touchscreen-enabled property was conceived by Guana et al. [13] to allow, or not, the player to interact with the game by using the touch gesture of Android mobile devices. We reinterpreted this property since in our case the user-interaction modality is based on Kinect. In particular, the exergame designer can set the touchscreen-enabled property to false if she wants to enable "soft" virtual on-screen hands, or true if she wants to enable "hard" virtual on-screen hands. The former will let the hands overlap the objects of the exergame so that the player can touch these objects by using the *closed-hand* gesture. The latter will let the hands interact with the objects when colliding with them—*i.e.,* the closed-hand gesture is disabled since it is not needed. Given the exploratory nature of our research, we decided not to modify the PhyDSL language (and the corresponding Eclipse plugin). That is to say that we planned to customize the PhyDSL language in case of promising results from our initial empirical assessment. The camera property manages the camera behavior during the exergame. It can be set to continuous or none. A continuous camera will continuously follow the main actor, moving smoothly through the exergame world and its frustum will always frame the scene respecting its boundaries. Conversely, a camera set to none will be static, and its frustum will always frame the center of the world.

The Activities definition section allows exergame designers to set up the activities of the exergame—the sub-phase **Definition of Exergame Activities** in Fig. 2. We show an example of the definition section for Alien Miner in Fig. 7. An activity is used to model an exergame event that is not directly triggered by the actions of the player. To define an activity, the exergame designer must specify the actor, frequency, angular velocity, linear speed, and position properties. The actor is the protagonist of the activity; the frequency indicates how often the event must be repeated; the angular velocity and linear speed specify the movement of the actor; and finally, the position defines where the actor has to appear.

The Scoring Rules definition section (see the example in Fig. 8) allows defining the scoring rules—the sub-phase **Definition of Exergame Scoring Rules** in Fig. 2. Three types of scoring rules are available: time-based rules; collision-based rules, and touch-based rules. Any scoring rule, regardless of its type, can trigger four different actions: update the exergame score, end the exergame, give sound and/or haptic feedback, and let an actor

```
design activities:

  appear activity meteoritesA (
    actor: meteorite
    frequency: 4.0
    angular velocity: slow
    linear speed: slow
    position: (25.0, 8.5) )

  appear activity meteoritesB (
    actor: meteorite
    frequency: 8.0
    angular velocity: slow
    linear speed: slow
    position: (40.0, 4.0) )

```

Fig. 7 Example of Activities definition section for AlienMiner

```
design scoring rules:
  rule: pointsUp collision of alien
        with emerald (
    points: 30
    game ends: false
    actor disappears: emerald
    haptic feedback: false )

  rule: pointsDown collision of
        alien with meteorite (
    points: -20
    game ends: false
    actor disappears: meteorite
    haptic feedback: false )

  rule: timeup timed 60 seconds (
    points: 0
    game ends: true
    haptic feedback: false )

  rule: collectDiamond touch of
        diamond (
    points: 20
    game ends: false
    actor disappears: diamond
    haptic feedback: false
    sound feedback: crash )
    ...
```

Fig. 8 Example of Scoring Rules definition section for AlienMiner

disappear. In PhyDSL$_K$ the haptic feedback property is ignored due to the interaction-modality based on Kinect.

Finally, in the Controls definition section (see the example in Fig. 9), exergame designers can set up the control system of the main actor—the sub-phase **Definition of Exergame Controls** in Fig. 2. It is possible to define moving and shooting controls. For both kinds of controls, Guana et al. [13] conceived the image and position properties to allow defining, respectively, the appearance and position of the corresponding buttons on the screen of Android mobile devices. In PhyDSL$_K$, the image and position properties are ignored since the interaction-modality is based on Kinect (*i.e.,* no button is visualized on the screen). To define moving controls, the moves property must be set: it specifies the force vector to apply to the main actor (*i.e.,* the intensity of the movement to be applied to the main actor as well as the direction of that movement). In PhyDSL$_K$, the x and y coordinates of the force vector allow also the recognition of the arm gestures, namely:

– *Right-arm* gestures, if x is greater than zero;
– *Left-arm* gestures, if x is less than zero;
– *Either-arm-up* gestures, if x is equal to zero and y is greater than zero.

For right-arm gestures, the angle defined by the force vector, with respect to the x axis, is used to recognize this kind of gestures. That is, a right-arm gesture is recognized when the angle defined by the right arm of the player, with respect to the x axis, is equal to or greater than the angle defined by the force vector. As for left-arm gestures, the recognition is similar but this time, instead of considering the angle defined by the right arm, the angle defined by the left arm is considered. As for either-arm-up gestures, the player must lift either arm over her head to allow the recognition of such a kind of gestures. We would like to recall that, for any gestures, the force vector defines the intensity of the movement to be applied to the main actor as well as the direction of that movement. To define shooting controls, the shoots and projectile properties must be set. The former specifies the force vector to be applied to the projectile, while the latter specifies which actor will be the projectile. In PhyDSL$_K$, shooting controls corresponds to *Lasso-hand* gestures. The x coordinate of the force

```
design main actor controls:
  control: jump (image: aBtn position: (0.0, 0.0) moves: up)
  control: right (image: aBtn position: (0.0, 0.0) moves: right)
  control: left (image: aBtn position: (0.0, 0.0) moves: left)
  control: jumpRight (image: aBtn position: (0.0, 0.0) moves: upRight)
  control: jumpLeft (image: aBtn position: (0.0, 0.0) moves: upLeft)
  control: fireRight (image: aBtn position: (0.0, 0.0) shoots:right projectile: shot)
  control: fireLeft (image: aBtn position: (0.0, 0.0) shoots:left projectile: shot)
```

Fig. 9 Example of Controls definition section for AlienMiner

vector allows determining if the corresponding lasso-hand gesture will be performed with the left hand (x less than zero), right hand (x greater than zero), or either hand (x equal to zero).

Once the exergame designer has provided the answers to the five gameplay concepts in a phy file, she can then export the exergame model—the sub-phase **Export of the Exergame Model** in Fig. 2. To that end, the PhyDSL plugin allows the exergame design to generate an *xmi* file, which stores the exergame model.

3.2 Code generation phase

When developing a code generator, three fundamental aspects must be considered: how much to generate, what to generate, how to generate [4]. A choice between a full generation or a partial one must be taken. Furthermore, the generated code should be maintainable and, at the same time, as concise as possible. To achieve such a (twofold) goal, reusable software components (*e.g.,* libraries or framework) can be used so that the generator limits itself to generate mostly *dynamic code* (*i.e.,* code that is different from a game to another one), while reusable software component encapsulates *static code* (*i.e.,* code that is in common among games). Bearing this in mind, we developed *(i)* PCAL to encapsulate the static code concerning Unity and *(ii)* inherited and tailored two scripts of the Kinect v2 asset to encapsulate the static code concerning Kinect—these scripts are referred, from here onwards, to as Kinect-interaction utility scripts. Thanks to these reusable software components, our code generator limits itself to perform four M2T transformations that lead to generate only four scripts, which depend on PCAL and Kinect-interaction utility scripts. The generated scripts follow:

- `MainScript`. This script manages all the logic of the exergame.
- `BodySourceView`. It visualizes the virtual hands of the player on the screen.
- `KinectGestures`. This script defines the gestures to interact with the exergame.
- `GestureListener`. It listens to the player when she executes the gestures defined through the `KinectGestures` script.

The latter three scripts are inherited and tailored from the Kinect v2 asset.

The code generator leverages the Acceleo transformation language to transform the exergame model into the above-mentioned scripts. Since Acceleo is a template-based transformation language, we defined a template for each script to be generated. Templates are made up of *meta-makers* and *text fragments*. Meta-makers are placeholders that have to be interpreted and evaluated while text fragments do not need to be interpreted. That is, at runtime, the template engine interprets meta markers and then replaces them with some text to produce the output files—in constant to text fragments that are reported as they are in the output files. In Fig. 10, we show an excerpt of the *mtl* file we used to define our four templates. An excerpt of the generated script is also shown.

To deploy the exergame, the generated scripts are added to a Unity template project, which encapsulates PCAL and Kinect-interaction utility scripts. A description of these two software components follows.

3.2.1 PCAL

The design of PCAL is based on the *pure code* approach [7]. It is an approach to design Unity-based games that minimizes the use of the Unity GUI editor while maximizing pure coding. PCAL consists of the following scripts:

Fig. 10 On top, an excerpt of the mtl file we used to define our templates. On bottom, an excerpt of the generated script

- `ResourcesManager`. The purpose of this script is to manage exergame resources such as loading sprites and audio clips.
- `ActorsManager`. This script replaces the standard Unity prefab system by creating custom prefabs of the actors defined in the exergame model, which are ready to be instantiated when needed. This scripts thus handle the creation and destruction of the actors; moreover, it provides the functionality to keep the main actor within the boundaries of the exergame world.
- `LayoutManager`. It allows locating the actors according to the specified coordinates.
- `EnvironmentManager`. This script manages the creation of the world scene. It provides methods for creating the background, the world borders, and the game camera along with the gravity setting.
- `ActivitiesManager`. This script provides the functionality for executing the activities defined in the exergame model.
- `ScoringRulesManager`. It allows handling time-based, collision-based, and touch-based scoring rules.
- `ControlsManager`. This script deals with managing the movements of the main actor, as well as shooting capabilities.
- `EventForwarder`. The purpose of this script is to manage the events of every game object in the scene, such as collision or update events.
- `EventForwarderManager`. This class deals with the event forwarder component attachment to game objects that need one. It saves references to the specific object in appropriate data structures, which allow future retrievals and action-performing.
- `GUIManager`. The purpose of this script is to manage the user interface. Thus, it exposes the methods for instantiating the canvas and the event systems, and the ones for creating and managing (*i.e.,* editing, showing, or hiding) GUI elements such as splash screens, timeboards, and scoreboards.

The use of PCAL leads to a very clean, neat, and readable `MainScript`, which mostly contains dynamic code, while the static code is mostly encapsulated in the library. This should allow having a positive impact of the maintainability of `MainScript`.

3.3 Kinect-interaction Utility Scripts

To allow the interaction with Kinect, the following two utility scripts are needed:

– `KinectManager`. This is the main Kinect-related script. It manages the communication between the Kinect sensor and the Unity-based exergame. It initializes the sensor, gets raw data from the cameras and processes them, gets body position and orientation, computes position of joints and their orientation starting from the main joints, computes confidence and state of hands, and detects the progress of gestures.
– `BodySourceManager`. The purpose of this script is to continuously retrieve the body data from the sensor and store them in an appropriate data structure.

4 Exergames

PhyDSL$_K$ allows end-users to develop different types of Kinect-based exergames, such as platform, shooter, puzzle, maze, and casual games. In the following of this section, we provide an overview of the capabilities of the system through three exergames we developed by using PhyDSL$_K$, namely: *KeepTheBalloons*, *PopTheBalloons*, and AlienMiner. These exergames have an increasing level of complexity. To develop KeepTheBalloons and PopTheBalloons, we took into account the requirements defined by the Department of Occupational Therapy of the University of Alberta. This is because both KeepTheBalloons and PopTheBalloons have been included in the Virtual Gym project, which has practical applications in upper-limb therapies for elder people [8]. As for AlienMiner, it was developed to include all features of PhyDSL$_K$ in a unique, fun, engaging exergame. AlienMiner is our Kinect-based exergame version of the game developed by Guana et al. for the Android platform [13]. Games were developed embracing criteria of Green IT [6].

4.1 KeepTheBalloons

KeepTheBalloons is a casual exergame (see Fig. 11). There is no main character to be moved. The exergame world is a bright and slightly cloudy sky. The goal of the exergame is to keep the balloons in the air as long as possible. All balloons come into view at the beginning of the exergame by falling from the sky. The player can interact with the balloons as

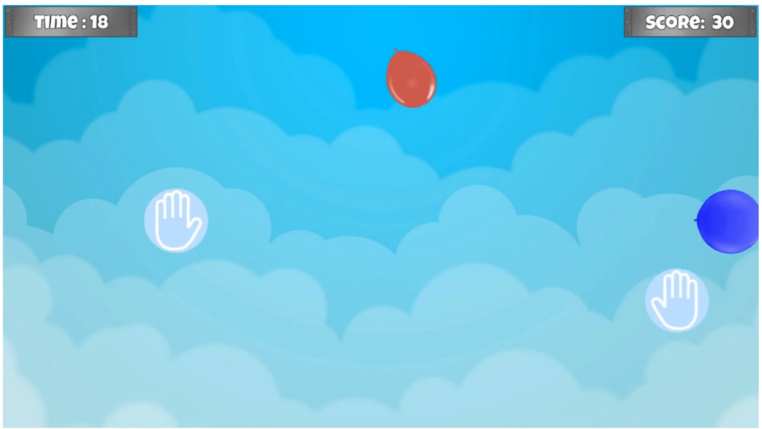

Fig. 11 A screenshot of KeepTheBalloons

she would do in real life, namely by pushing them with more or less strength based on how close the balloons are to the ground. The score is incremented every five seconds based on the number of balloons in the air. In particular, each balloon in the air provides 5 points every 5 seconds. Three different difficulty levels are available, namely: easy, medium, and hard. The higher the difficulty level is, the higher the number of balloons that come into view during the exergame is, as well as the greater the diversity of color and shape of the balloons are. It is worth mentioning that, at the current stage of this research, each level of the exergame actually corresponds to a new exergame. The exergame ends when one of the balloons reaches the floor or after two minutes. KeepTheBalloons was developed for use in upper-limb therapies. In particular, it aims to let players (*i.e.,* elder people) familiarise themselves with the exergaming world by trying to emulate natural gestures, such as lifting an arm to push a balloon, along with any possible natural movement that makes the exergame challenging for arms.

4.2 PopTheBalloons

PopTheBalloons is a casual exergame as well (see Fig. 12). There is no main character to be moved. Similar to KeepTheBalloons, the exergame world is a bright and slightly cloudy sky. The balloons come into view during the exergame by falling from the sky. The goal of the exergame is to pop the balloons before they reach the floor. The player can pop the balloons by performing a closed-hand gesture when the corresponding virtual hand overlaps the balloon. Each popped balloon lets the player score be incremented by five points. Conversely, the score does not change when balloons reach the floor so popping by themselves. This choice is to avoid discouraging low-performing players. Three different difficulty levels are available, namely: easy, medium, and hard. The higher the difficulty level, the higher the number of balloons that come into view during the exergame, as well as the greater the diversity of colors and shapes of the balloons. The exergame ends after two minutes. PopTheBalloons was developed to let players (*e.g.,* elder people) move their arms up, down, and to the sides, as well as lead them to bend their torso to reach and pop balloons that would be not accessible otherwise.

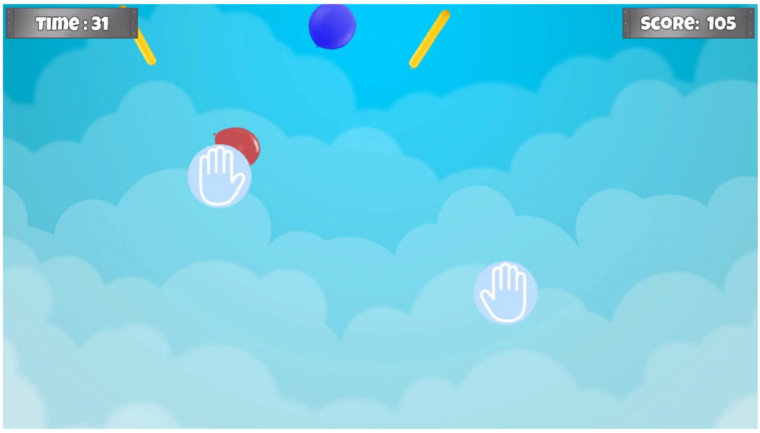

Fig. 12 A screenshot of PopTheBalloons

4.3 AlienMiner

AlienMiner is a 2D game initially developed by Guana et al. [13]. We upgraded the game with a new Kinect-based interaction and new shooting capabilities. The exergame starts with an alien (*i.e.,* the main actor) located on a foreign planet. To complete a level, the player has to drive the alien towards the spatial portal. The planet is full of dangers and obstacles, along with precious gems that the player should collect to increase the score. The controls are based on arm movements (*i.e.,* arm gestures). The player can collect gems in different ways. For example, emeralds are collected when reached by the alien—for each collected emerald, the score of the player is increased by 30 points. Diamonds are, instead, collected when the player closes her hand (*i.e.,* closed-hand gesture) while the respective virtual hand overlaps the gem on the screen. In this case, the score is incremented by 20 points. The player should avoid collisions with meteorites. When this happens, the player is penalized by 20 points. Meteorites can be shot by the player. In particular, the player can shoot fireballs towards meteorites by performing the Lasso-hand gesture with her left or right hand. The exergame ends when the spatial portal is reached by the alien or after two minutes. Some screenshots of AlienMiner are shown in Fig. 13.

5 Empirical study

In this section, we present the empirical study we carried out to preliminary evaluate PhyDSL_K, along with the obtained results.

5.1 Study planning and execution

We planned and executed our empirical study by bearing in mind the guidelines by Wohlin et al. [24].

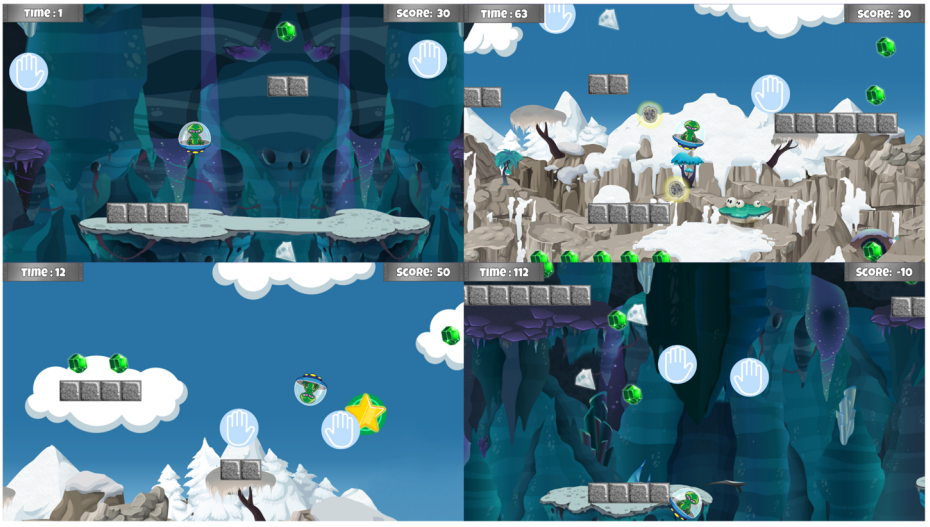

Fig. 13 Some screenshots of AlienMiner

5.1.1 Study goal

We define the goal of our empirical study by using the Goal/Question/Metric template [3] as follows:

> *Analyze* PhyDSL$_K$ *for the purpose of* evaluating it *with respect to* the effort to develop exergames and its usability regardless of the exergame complexity *from the point of view of* researchers and end-users *in the context of* undergraduate students.

Based on the above-mentioned goal, we defined and investigated the following Research Questions (RQs):

RQ1. What is the effort to develop exergames by using PhyDSL$_K$ regardless of the exergame complexity?
RQ2. What is the usability of PhyDSL$_K$ regardless of the exergame complexity?

5.1.2 Study participants

We asked 14 undergraduate students, taking the Software Engineering course at the University of Bari (Italy) and Polytechnic of Bari (Italy), to voluntarily take part in the empirical study. Based on a pre-questionnaire the participants had filled in before the study took place, no one was familiar with either Unity or Kinect. Furthermore, they had no experience with game development and did not know the PhyDSL language. Therefore, we can assume the participants as representative of end-users (*e.g.,* therapists) that do not have experience in game development but want to define the gameplay of exergames and then develop the exergames.

5.1.3 Variable selection

The dependent variables we used to answer, respectively, RQ1 and RQ2 are:

- **Effort.** It is measured through two metrics: *(i)* time (in hours) consumed by a participant to develop an exergame by using PhyDSL$_K$; and *(ii)* LOC (Lines of Code, excluded blank and commented lines) of the phy file a participant wrote to develop an exergame. For both metrics, the greater they are, the greater the effort to develop an exergame is.
- **Usability.** We used the System Usability Scale (SUS) questionnaire [5] to compute the SUS Score, which is a metric of usability. This questionnaire consists of ten statements, each of which can be rated on a 5-point scale—*i.e.,* from 1 ("I strongly disagree") to 5 ("I strongly agree"). The questionnaire alternates five positive statements (*e.g.,* "I think that I would like to use this system frequently") to five negative statements (*e.g.,* "I found the system unnecessarily complex"). Let Score$_i$ be the score (from 1 to 5) of the i-th statement, the SUS Score is computed according to the following formula [5]:

$$SUS\ Score = 2.5 * \sum_{i=1}^{10} \begin{cases} Score_i - 1 & \text{if the i-th statement is positive} \\ 5 - Score_i & \text{if the i-th statement is negative} \end{cases}$$

  The SUS score ranges in [0, 100]. The greater a SUS score is, the better the usability of PhyDSL$_K$ is. A SUS score greater than or equal to 70 indicates a good usability [2].

The participants in the study were asked to develop, by using PhyDSL$_K$, three exergames with increasing complexity, namely: KeepTheBalloons, PopTheBalloons, and AlienMiner.

Therefore, **Exergame** is an independent variable that assumes three values (*i.e.,* KeepTheBalloons, PopTheBalloons, and AlienMiner).

5.1.4 Study design

The participants took part in four sessions, which were held in the same week. The first one was a training session conceived to let participants became familiar with PhyDSL$_K$. The remaining ones were the actual experimental sessions in which each participant was asked to define the gameplay of three exergames—*i.e.,* one experimental session for each exergame to be developed.

The training session consisted of two 50-minute blocks divided by a 10-minute break. The first part presented PhyDSL$_K$ while the second part consisted of a tutorial on the use of PhyDSL$_K$ where each participant familiarized with the system by using a sample exergame.

In each experimental session, a participant was asked to perform an experimental task; namely, she was asked to develop an exergame by using PhyDSL$_K$. In particular, she had to: *(i)* create the exergame model encapsulating the gameplay for the assigned exergame; *(ii)* generate the four scripts from the exergame model; and *(iii)* deploy the exergame to verify it worked as expected. To do so, the participant was provided with the specifications of the exergame (see Section 5.1.5), the folder of multimedia resources (*e.g.,* it contains the background images), and the documentation of PhyDSL$_K$.

The exergames were assigned to the participants from the least to the most complex. In particular, KeepTheBalloons was the first task, PopTheBalloons was the second one, and AlienMiner was the last one. To mitigate the fatigue effect, the experimental sessions took place on different days.

For each experimental task, we keep track of the time each participant consumed to develop the exergame. After performing the experimental task, each participant was asked to rate the usability of PhyDSL$_K$ by filling in the SUS questionnaire [5]. Finally, we extracted the LOC from the phy file each participant wrote to develop the exergame.

5.1.5 Experimental tasks

A description of each task, along with the specifications of the corresponding exergame we supplied to each participant, follows.

KeepTheBalloons The goal of the task was to investigate how the participants addressed a relatively simple exergame scenario, which did not involve all features of PhyDSL$_K$. The specifications of KeepTheBalloons follow:

- The goal is to let a player move her arms/hands in order to push and keep some balloons in the air as long as possible.
- All balloons must come into view at the beginning of the exergame by falling from the sky.
- The background must consist of one image only (so the size of the background is 1920x1080 pixels).
- The camera must be static.
- The exergame must last at most two minutes.
- The exergame must end when a balloon reaches the ground.
- Time is the metric that governs the score of the player (*i.e.,* the higher the exergame time, the higher the score).

– The choice of further implementation details is left to each participant (*e.g.,* number, color, or size of the balloons).

PopTheBalloons The task required the participants to define the gameplay of a more challenging exergame, which involved more PhyDSL$_K$ features as compared to KeepTheBalloons.

– The goal is to let a player move her arms/hands and bend her torso to pop as many balloons as possible.
– Balloons must come into view dynamically during the exergame (*i.e.,* new balloons must come into view with time).
– The player can pop the balloons by means of the closed-hand gesture.
– Balloons must pop when they reach the ground.
– The background must consist of one image only (so the size of the background is 1920x1080 pixels)
– The camera must be static.
– The number of popped balloons governs the score of the player (*i.e.,* the higher the number of popped balloons, the higher the score).
– The exergame must last two minutes.
– The choice of further implementation details is left to each participant (*e.g.,* number, color, or size of the balloons).

AlienMiner This task aimed to investigate how the participants addressed a complicated exergame scenario that involved all PhyDSL$_K$ features.

– The goal is to let a player move her arms/hands so that the alien (*i.e.,* the main actor) can reach the spatial portal.
– The alien is located on a planet with obstacles (*i.e.,* bricks), meteorites, and gems (*i.e.,* emeralds and diamonds).
– The player can control the alien through specific arm gestures: left/right arm extended outwards (*i.e.,* horizontally) to move the alien towards the left/right side; left/right arm extended upwards to move the alien up; left/right arm extended diagonally upwards to move the alien up-left/up-right.
– The player can let the alien shoot fireballs by using hand gestures: lasso-hand gesture performed with the left hand to shoot towards left; lasso-hand gesture performed with the right hand to shoot towards right.
– Bricks, emeralds, diamonds, and spatial portal must be located statically.
– Meteorites must come into view dynamically during the exergame.
– The player can collect emeralds when the alien collides with them.
– The player can collect diamonds by performing a hand-closing gesture.
– Meteorites disappear when colliding with each other or with the alien.
– Fireballs can destroy the meteorites only.
– The background must consist of multiple images.
– The camera must be dynamic (*i.e.,* it must follow the main actor).
– The number of collected gems and collisions with meteorites governs the score of the player. In particular, the former let the score increase, while the latter lets the score decrease).
– The exergame must last two minutes.
– The exergame must end when the alien reaches the spatial portal.

– The choice of further implementation details is left to each participant (*e.g.,* position of bricks or gems).

5.1.6 Data analyses

We analyzed the data by using the *R* environment for statistical computing.[2] We computed descriptive statistics (*e.g.,* mean, median, *etc.*) and performed exploratory data analyses (*e.g.,* boxplots) to summarize the distributions of the metrics. We then tested whether the exergames (represented by the Exergame independent variable) had affected the metrics. Given the data dependency due to repeated measures—we measured each participant three times—, we used either the Linear Mixed Model (LMM) by modeling the participants as random independent variable, or its non-parametric alternative, namely the Friedman test [23]. In particular, we used the former, if the assumption of normality underlying the LMM was satisfied, while we used the latter, otherwise. To check the normality assumption, we applied the Shapiro-Wilk test. For any test of statistical hypotheses, we fixed the significance level, α, at 0.05. This implies that the effect of the exergame was deemed significant if the p-value returned by the test of statistical inference (*e.g.,* LMM) was lower than α.

5.1.7 Threats to validity

Although we addressed as many threats to validity as possible, some of them are unavoidable. We discuss the threats to validity that might affect our empirical study, and its results, based on internal, external, construct, and conclusion validity (if any) [24].

Internal Validity. It concerns factors that might influence the obtained results without researchers' knowledge [24]. The participants voluntarily took part in the empirical study. This might influence the results since volunteers are generally more motivated than the entire population (*i.e.,* threat of *selection*) [24]. A threat of *maturation* might affect the results. That is, some participants might react differently as time passes (*e.g.,* fatigue effect). To mitigated this threat, we executed each of the experimental tasks in different days.

Construct Validity It refers to how well the study results support the theory behind the study. Using a single metric or performing a single measurement to quantify an independent variable might imply a threat of *mono-method bias*. While we measured the development effort by using two metrics (*i.e.,* time and LOC) and performing three measurements for each metric (and each participant), we measured the usability of PhyDSL_K by using a single metric (*i.e.,* SUS score). However, for each participant, we performed three measurements for the SUS score to mitigate such a threat.

External Validity. It refers to the degree to which researchers can generalize the results of a study to other participants, conditions, times, and places. We selected the participants as being a representative sample of end-users that have no experience with game development, such as therapists. As so, we selected students who attended the Software Engineering course without experience in game development. The use of students as participants fits for studies whose nature is *theory testing* [24] (just like ours). Furthermore, the PhyDSL language, which PhyDSL_K is based on, has been conceived for non-programming experts [13]

[2] https://www.r-project.org/

and can be learned in short time—our participants were capable of using PhyDSL$_K$ after a training of about 100 minutes. Having said that, we are aware that this represents a threat to the generalizability of the results (*i.e.,* threat of *interaction of selection and treatment*) since, for instance, therapists are more knowledgeable of the application domain (*i.e.,* rehabilitation exergames) and less expert in programming than the students involved in the study. Given the above-mentioned considerations, we believe the study results can be safely accepted although further studies with domain experts would strengthen the external validity of the results. Furthermore, the results of our study could help us to motivate more expensive studies (*e.g.,* studies with therapists, which are more difficult to recruit than students). Finally, the used exergames might affect the results (*interaction of setting and treatment*). In particular, while PopTheBalloons and KeepTheBalloons have been applied in elder people therapies within the Virtual Gym project [8], AlienMiner has not been applied yet in that application field. We chose AlienMiner to introduce a more complex exergame than the other two, which also included all features of PhyDSL$_K$, and implemented a game for other devices (*i.e.,* Android phones) [13].

5.2 Results

In Table 2, we report the values of min, mean, median, max, and Standard Deviation (SD) for each metric by considering each exergame and all exergames.

5.2.1 Answering RQ1

By looking at Table 2, we can notice that it took roughly the same time to develop the three exergames despite they differed in complexity. In particular, all participants spent one hour per exergame with the following three exceptions: just one participant spent two hours developing KeepTheBalloons and just two participants spent, respectively, two and three hours developing AlienMiner. This trend seems to suggest that the exergame, as well as

Table 2 Descriptive statistics for each metric when considering each exergame and all exergames

Metric	Statistic	KeepTheBalloons	PopTheBalloons	AlienMiner	All
Time	Min	1	1	1	1
	Mean	1.071	1	1.214	1.095
	Median	1	1	1	1
	Max	2	1	3	3
	SD	0.267	0	0.579	0.37
LOC	Min	67	65	205	65
	Mean	105.5	126.286	249.571	160.452
	Median	104	135.5	251	138.5
	Max	138	172	277	277
	SD	21.929	31.517	22.149	69.027
SUS Score	Min	72.5	50	70	50
	Mean	81.964	80	79.286	80.417
	Median	82.5	81.25	78.75	80
	Max	97.5	100	87.5	100
	SD	6.443	15.128	5.041	9.752

its complexity, did not significantly affect the hours needed to develop that exergame. To confirm such a result, we ran a test of statistical inference. In particular, we could not apply LMM because the normality assumption was not satisfied as suggested by the Shapiro-Wilk test (p-value $\approx$ 0). Therefore, we ran the Friedman test, which returned a p-value (0.156) greater than 0.05 so confirming that the exergame, as well as its complexity, did not significantly affect the hours needed to develop the exergame.

On the other hand, we can notice (see Table 2) that the LOC tends to increase when passing from the least to the most complex exergame. For example, the LOC are on average 105.5 for KeepTheBalloons, and increase up to 126.286 and 249.571 for PopTheBalloons and AlienMiner, respectively. By looking at the boxplots in Fig. 14, such an increasing trend is more evident. In fact, the boxes for KeepTheBalloons and PopTheBalloons mostly overlap one another, although the one for PopTheBalloons is higher than the one for KeepTheBalloons, and the box for AlienMiner is higher than the others without overlapping them. We then analyzed the effect of the exergame on the number of LOC the participants wrote. To this end, we used the LMM since the assumption of normality was satisfied as suggested by the Shapiro-Wilk test (0.21). The p-value returned by the LMM was approximately equal to 0 so indicating that there is a statistically significant difference between the exergames with respect to the LOC.

The above-mentioned results seem promising since our empirical study shows that people without experience in game development, but with a 100-minute training on the use of PhyDSL$_K$, can develop exergames with different complexity levels in one hour, on average. As for the amount of code needed to develop exergames, it can vary when moving from a simpler to a more complex exergame. Nevertheless, one working hour seems to be enough to address a more complex exergame like AlienMiner, which requires the gameplay designer to write, on average, 250 LOC.

5.2.2 Answering RQ2

As shown in Table 2, the participants tended to rate the usability of PhyDSL$_K$ as good. For example, whatever the exergame was, the SUS score was on average not inferior to 70 so indicating a good usability for PhyDSL$_K$. The boxplots in Fig. 15 confirm that most of the

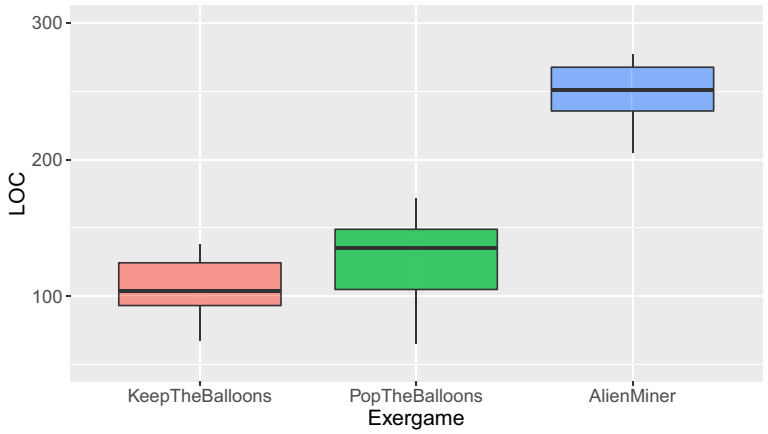

Fig. 14 Boxplots summarizing the distributions of LOC for each exergame

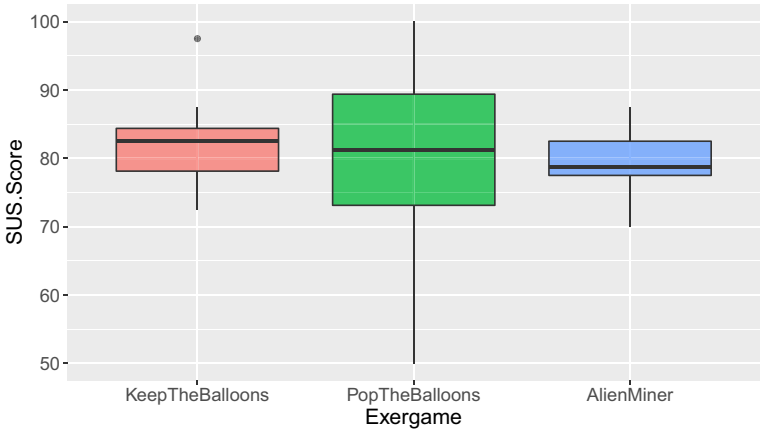

Fig. 15 Boxplots summarizing the distributions of SUS scores for each exergame

participants rated the usability of PhyDSL$_K$ as good. In particular, we can notice that the distributions of the SUS scores for KeepTheBalloons and AlienMiner are entirely above 70. As for the distribution of the SUS scores for PopTheBalloons, it indicates that most of the SUS scores are above 70. We can also observe that the boxes overlap one another. This seems to indicate that there is no significant difference due to the exergame to be addressed. We used the Friedman test to confirm this observation because the data did not satisfy the assumptions of normality behind the use of the LMM—the Shapiro-Wilk test returned a p-value equal to 0.045. The p-value (0.465) returned by the Friedman test did not suggest a significant difference in the SUS scores due to the exergame to be addressed.

Summing up, our empirical investigation shows that people with no experience in game development consider PhyDSL$_K$ usable; furthermore, the usability of PhyDSL$_K$ does not depend on the exergame to be addressed and its complexity.

6 Conclusion and future work

Exergames are complex software systems, demanding expertise in exercise principles, user-interaction models, and challenging hardware/software development. To mitigate this complexity and enable the prototyping and development of exergames by developers with relatively little programming experience, we developed PhyDSL$_K$, a model-driven software toolkit for the user-centered development of exergames.

We evaluated PhyDSL$_K$ with an empirical study designed to assess its usability and the degree to which it simplifies development effort. Our results are promising and confirm the appropriateness of PhyDSL$_K$ as a framework to support end-users in creating exergames without needing to be knowledgeable with either Unity and Kinect or have specific programming experience. Our results point out that, even if the complexity of the exergames increases, the development effort remains relatively the same. Moreover, we provide evidence that end-users consider the PhyDSL$_K$ framework highly usable, given the average SUS score achieved (*i.e.,* 80/100), regardless of the complexity of the exergame to develop. These initial results are quite promising and encourage us to further investigate PhyDSL$_K$.

Our future research agenda includes:

- Replicating our preliminary study to extend the external validity of the obtained results. In particular, we are going to conduct in-vivo replications to assess development effort and usability when domain experts, such as therapists, had to create exergames for their patients through PhyDSL_K. Although such a kind of studies is expensive because therapists are more difficult to recruit (as compared to students), we believe that the promising results presented in this paper can foster the participation of therapists in our replications.
- Upgrading the DSL (*i.e.,* PhyDSL), by introducing new types of activities (*e.g.,* disappear activities) and scoring rules (*e.g.,* frequency-based rules) in order to match domain experts' needs more accurately, as well as expanding the control system to include lower-limb gesture recognition so that experts can design a more complete exercise regimen and create whole-body therapeutic exergames.
- Developing new exergames for rehabilitation purposes and evaluating them with potential patients.

Acknowledgements This work is the result of a research collaboration between SERLAB (University of Bari – Department of Informatics), SSRG (University of Alberta, Computer Science Department) and (University of Alberta, Occupational Therapy Department). During his research stay Francesco Cagnetta received a grant from UofA. Drs. Noellanah Neubauer and Lili Liu have provided their expertise and feedback to the development of VirtualGym, which has also been instrumental for this work. We thank all the students that volunteered to be part of the empirical study.

Funding Open access funding provided by Università degli Studi di Bari Aldo Moro within the CRUI-CARE Agreement. This study has been partially funded by the Project "HEVOLUS+" (Cod.OH4JBL3) funded by FSC - APQ Sviluppo Locale 2007–2013-Titolo II-Capo 2 - Regione Puglia.

References

1. Averell E, Knox D (2019) A rhythm-based game for stroke rehabilitation. In: Proceedings of International Conference on Immersive and Interactive Audio. Audio Engineering Society
2. Bangor A, Kortum PT, Miller JT (2008) An empirical evaluation of the system usability scale. Int J Human-Comput Interact 24(6):574–594. https://doi.org/10.1080/10447310802205776
3. Basili VR, Rombach HD (1988) The tame project: towards improvement-oriented software environments. IEEE Trans Softw Eng 14(6):758–773. https://doi.org/10.1109/32.6156
4. Brambilla M, Cabot J, Wimmer M (2017) Model-driven software engineering in practice, nd edn. Morgan & Claypool Publishers
5. Brooke J (1996) Usability evaluation in industry. CRC Press
6. David Patón-Romero J, Baldassarre MT, Piattini M, de Guzmán IGR (2017) A governance and management framework for green it. Sustainability 9:1761. https://doi.org/10.3390/su9101761
7. Dunstan J (2015) A pure code approach to unity app code design. https://jacksondunstan.com/articles/2914
8. Fernandez-Cervantes V, Neubauer N, Hunter B, Stroulia E, Liu L (2018) Virtualgym: A kinect-based system for seniors exercising at home. Entertain Comput 27:60–72. https://doi.org/10.1016/j.entcom.2018.04.001

9. Fernandez-Cervantes V, Stroulia E, Hunter B (2016) A grammar-based framework for rehabilitation exergames. In: Proceedings of International Conference on Entertainment Computing. Springer, pp 38–50
10. France R, Rumpe B (2007) Model-driven development of complex software: A research roadmap. In: Proceedings of Future of Software Engineering, pp 37–54
11. Galna B, Barry G, Jackson D, Mhiripiri D, Olivier P, Rochester L (2014) Accuracy of the microsoft kinect sensor for measuring movement in people with parkinson's disease. Gait Post 39(4):1062–1068
12. Gao Z, Lee JE, Pope Z, Zhang D (2016) Effect of active videogames on underserved children's classroom behaviors, effort, and fitness. Games Health J 5(5):318–324. https://doi.org/10.1089/g4h.2016.0049
13. Guana V, Stroulia E, Nguyen V (2015) Building a game engine: A tale of modern model-driven engineering. In: Proceedings of International Workshop on Games and Software Engineering. IEEE, pp 15–21
14. Hardy S, Dutz T, Wiemeyer J, Göbel S, Steinmetz R (2015) Framework for personalized and adaptive game-based training programs in health sport. Multimed Tools Appl 74(14):5289–5311. https://doi.org/10.1007/s11042-014-2009-z
15. Lange B, Chang C, Suma E, Newman B, Rizzo AS, Bolas M (2011) Development and evaluation of low cost game-based balance rehabilitation tool using the microsoft kinect sensor. In: Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp 1831–1834
16. Li J, Erdt M, Lee JCB, Vijayakumar H, Robert C, Theng Y (2018) Designing a digital fitness game system for older adults in community settings. In: Proocedings of International Conference on Cyberworlds, pp 296–299
17. Mocanu I, Marian C, Rusu L, Arba R (2016) A kinect based adaptive exergame. In: Proccedings of International Conference on Intelligent Computer Communication and Processing, pp 117–124
18. Ofli F, Kurillo G, Obdržálek S, Bajcsy R, Jimison HB, Pavel M (2016) Design and evaluation of an interactive exercise coaching system for older adults: Lessons learned. IEEE J Biomed Health Inf 20(1):201–212. https://doi.org/10.1109/JBHI.2015.2391671
19. Pastor I, Hayes HA, Bamberg SJM (2012) A feasibility study of an upper limb rehabilitation system using kinect and computer games. In: Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp 1286–1289
20. Portes CG, Lacave. C, Molina. AI, Vallejo. D, Sánchez-Sobrino. S (2020) Personalising exergames for the physical rehabilitation of children affected by spine pain. In: Proceedings of International Conference on Enterprise Information Systems. SciTePress, pp 533–543
21. Sáenz-de-Urturi Z, García Zapirain B, Méndez Zorrilla A (2015) Elderly user experience to improve a kinect-based game playability. Behav Inf Technol 34(11):1040–1051. https://doi.org/10.1080/0144929X.2015.1077889
22. Staiano AE, Calvert SL (2011) The promise of exergames as tools to measure physical health. Entertain Comput 2(1):17–21
23. Vegas S, Apa C, Juristo N (2016) Crossover designs in software engineering experiments: Benefits and perils. IEEE Trans Softw Eng 42(2):120–135
24. Wohlin C, Runeson P, Hst M, Ohlsson MC, Regnell B, Wessln A (2012) Experimentation in software engineering. Springer