Education and Information Technologies (2023) 28:283–302
https://doi.org/10.1007/s10639-022-11158-6

Game design and didactic transposition of knowledge. The case of progo, a game dedicated to learning object-oriented programming

Fahima Djelil[1] · Eric Sanchez[2]

Received: 2 December 2021 / Accepted: 1 June 2022 / Published online: 29 June 2022

Abstract
Game based-learning have been widely promoted to overcome the difficulties encountered by beginners to learn programming. However, there are many issues to address for the implementation of game-based learning. Indeed, game-based learning is not limited to adding game elements such as rewards to a learning situation, but it rather consists of transforming the learning situation so that it becomes playful. This work contributes to computer science education research, especially to game design for learning programming. We design a novel environment dedicated to learning object-oriented programming for beginners called Progo. It is based on a metaphor of a three-dimensional (3D) construction and animation game. We present an *a priori* analysis of the Progo environment on the basis of a didactic transposition framework. The framework highlights the ludicisation and metaphorisation process by which educational content is integrated into the game. This allows for the analysis of the transformation of the computing knowledge by the game design, and to verify whether analogies are maintained between the knowledge and what the learner should experience through play. This work contributes to a framework for the integration of educational content during learning game design.

Keywords Game design · Didactic transposition · Metaphors · Ludicisation · Programming Learning · Progo

✉ Fahima Djelil
fahima.djelil@imt-atlantique.fr

Eric Sanchez
eric.sanchez@unige.ch

1 IMT Atlantique, Lab-STICC, UMR CNRS 6285, Brest 29238, France

2 TECFA, University of Geneva, 40 Bd du Pont d'Arve, Geneva 1211, Switzerland

1 Introduction

Learning and teaching programming to beginners is proven to be difficult, it is considered as one of the major challenges in computer science education (Medeiros et al., 2018; Bennedsen, 2008; Piteira & Costa, 2013). Recent works focused on teaching and learning object-oriented programming, which is one of the most difficult paradigm for beginners (Abidin & Zawawi, 2020; Abbasi et al., 2017; Seng & Yatim, 2014; Keung et al., 2018). The main difficulty that students face, lies in abstract basic concepts, and they struggle to understand more complex concepts, when they cannot grasp the basics.

Previous research emphasises that a curriculum may focus on the use of concrete and visible objects, and games to motivate and engage students to develop their computing competences (Webb et al., 2017). Many works on introductory programming teaching are directed towards the use of microworlds (Michaelson, 2018; Woei et al., 2015; Costa & Miranda, 2017; Yukselturk & Altiok, 2017). Microworlds are visual and interactive environments, that aim to help students to understand abstract concepts through play and visible objects (Papert, 2020). For instance, programming microworlds such as Scratch (Resnick et al., 2009), Alice (Cooper et al., 2000) and Greenfoot (Kölling, 2009) allow students to manipulate visible and concrete objects and create their own game scenarios, leading to playful and engaging learning activities.

Game-based learning consists of offering the learner to participate in a playful situation, and to develop reflexivity towards his learning experience (Sanchez, 2019). The design of such playful situation involves transforming a learning situation in a way that allows the learner to adopt a playful attitude (Sanchez, 2019). This process, called ludicisation (Genvo, 2011) is a common implementation of programming microworlds, in which the concept of metaphor is very apparent (Djelil et al., 2016). Indeed, metaphors allow to describe abstract concepts in a more comprehensible and concrete way for beginners, by providing analogies from familiar domains (Lakoff & Johnson, 2008). Through these two properties, ludicisation and metaphorisation, microworlds provide learning experiences that aim, on the one hand, to make programming concepts concrete to be more easily graspable by beginners, and on the other hand, to make programming learning attractive to capture students' attention and increase their motivation and interest (Papert, 2020; Moskal et al., 2004).

In this paper, ludicisation and metaphorisation are approached from the perspective of the didactic transposition framework (Colomb, 1986), as a process by which academic knowledge is transformed to become learning objects, allowing students to learn through experience. Thus, didactic transposition serves as a framework for game design. In this framework, ludicisation implies metaphorisation to reshape the concepts to be learned. This framework has been displayed outside the French-speaking community, and applied for the teaching of many disciplines (Bosch & Gascón, 2006). However, existing works addressing the question of didactic transposition relate mainly to the domain of mathematics and sciences and rarely to the domain of computer science (Hazzan et al., 2010). Indeed, in the didactics of computer science, existing work (Orange, 1990) focuses more on the analysis of learners' skills and difficulties, than on the transposition of knowledge for designing learning situations.

We, therefore, contribute with this paper to the growing research in the field of computer science education, raising questions on how to make programming learning attractive and effective to beginners. Our work relies on the introduction of object-oriented principles for beginners and on ludicisation for the design of a new environment called Progo. We refer to beginners by novice students that are introduced to basic concepts of object-oriented programming instead of advanced concepts. Based on the didactic transposition framework, we analyse the Progo game design, to understand how the object-oriented programming concepts are transformed in a playful situation through a construction game metaphor, and how the meaning of these concepts is maintained after transformation. Thus, the research questions we address are the following: 1) How computing knowledge is transformed by game design ? 2) To what extent analogical relations are maintained between the computing knowledge and the concrete and visible objects ?

The purpose of this paper is threefold, both theoretical, empirical and policy relevant. First, we present a theoretical model of didactic transposition for the design of learning situations, leading students to learn through discovery and play. Then, we apply this model to the case of Progo. This empirical work consists of an *a priori* analysis of the game design, and more specifically, of how teaching knowledge is integrated into the game. Finally, we aim to bring a broader contribution in terms of implications for learning game design in the filed of computer science education.

In the following sections, we first trace previous work on the learning design inherent to the objects-first approach for the introduction of object-oriented principles for beginners, as well as on game-based programming learning. We describe the didactic transposition model for game design which is based on the concepts of ludicisation and metaphorisation. Then, we describe our method used to analyse and discuss the Progo game with the lenses of the ludicisation framework. The last sections emphasize the limitations, conclusions and implications of this work.

2 Related work

2.1 Objects-first approach for teaching object-oriented programming to beginners

Programming is what allows a sequence of operations to be executed on a computer. The goal of any program is to compute and return valid and reliable results. A program is defined as a sequence of actions that a computer must perform in a finite time to resolve a problem (Dabancourt, 2008). These operations are called instructions and are translated into a programming language. The way a program is constructed constitutes a programming style or a programming paradigm.

The concepts of object and class are the basic concepts that govern the object-oriented paradigm. A class is often defined as a structure representing a mould or a template for the object, while the object is an embodiment, a reproducible copy of its class (Bersini, 2017). Different approaches exist for teaching programming and the object-oriented paradigm (Roberts & et al., 2001). Objects-first approach focuses on the fundamentals of object-oriented programming and design at the very beginning of curriculum. It is one of the most quoted approach in the literature, reflecting its interest in practice and its potential to overcome learning and teaching difficulties (Bennedsen, 2008; Woei et al., 2015; Michaelson, 2018; Krugel & Hubwieser, 2018). For instance, searching ”objects-first approach” in the ACM Digital Library returns 561,358 results (October, 2021).

Many authors described their experiences for the introduction of object-oriented courses with respect to the object-first approach. The approach described by (Woodworth & Dann, 1999) and (Adams & Frens, 2003), guides the learner for the design and the use of programming abstractions, by modelling properties of objects used in a previous problem-solving step. (Buck & Stucki, 2000) argue on an approach that allows the students to start by changing the computer code and later, designing parts of the object-oriented system. Programming language concepts are introduced as needed during the problem-solving tasks. Similarly, (Becker, 2001) describes a learning scenario that consists of starting by instantiating and using objects and then extending existing classes. Programming language fundamentals are introduced progressively. This learning scenario includes the use of the microworld Karel (Xinogalos et al., 2006). (Kölling & Rosenberg, 2001) suggest a set of guidelines for teaching introductory object-oriented programming: Starting with objects from the beginning instead of a small program in an imperative paradigm, modifying existing code, reading a well-structured code, etc. This approach is implemented in the Greenfoot microworld (Kolling, 2015).

These experiences are very similar as they do not focus on programming languages at the beginning of teaching, but rather on Object-Oriented concepts, through problem-solving tasks using significant learning environments, such as microworlds. This approach was further conceptualised on the basis of analysis of more than 200 contents in the literature (Bennedsen & Schulte, 2007), allowing the definition of three steps through which it is implemented in practice:

1. Using objects: the learner uses objects defined before programming classes. The focus is on the use of objects before their implementation (objects interaction introduced before methods implementation). Once the learner masters the concept of object, he moves onto the concept of classes.
2. Creating classes: the learner defines and implements classes and creates instances of classes (data fields and methods). The focus is on writing and using classes before algorithms. The concept of a class is often approached in a concrete and creative programming way.
3. Concepts: the learner learns general principles of the object-oriented paradigm, through the creation of models. The focus is on the conceptual aspects of object-orientation. The objective is to learn to model a real world as objects and to map these objects to classes of code.

It is worthy to consider the learning design (Brousseau, 2006) that defines the objects-first approach to introduce the object-oriented paradigm to beginners. Objects-first approach allows for the introduction of object-oriented programming principles through the embedding of three categories of concepts (Djelil et al., 2020). At each level, the objective is to help the beginner to focuse on a category of concepts, while the successive ones are temporally hidden. The categories are embedded since they encapsulate object-oriented concepts that are interdependent (objects, classes and design principles) (Fig. 1). As a result, beginners are expected to progressively acquire prerequisites before handling complex concepts, when they are involved in modelling and coding object-oriented programs for problem-solving.

2.2 Game-based programming learning

According to (Plass et al., 2015), game-based learning depends on the alignment between the game characteristics and learning outcomes. However this issue is relatively unaddressed. A meta-analysis published in 2016, highlighted only 8 articles out of 69 that explicitly addressed this issue (Ke, 2016).

Yet, as soon as 2007, (Habgood, 2007) distinguished between games that are described as extrinsic games, for which the game content and academic exercises alternate (the game then appears to be a reward for having succeeded in the exercise), and intrinsic games, for which the targeted knowledge is necessary to deal with the objectives of the game. Thus, using a game-based approach to teach programming leads to integrating educational content with playful aspects. This is what is referred to intrinsic metaphor for a successful integration (Fabricatore, 2000).

In the field of computer science education, programming microworlds are designed as intrinsic games, since they are based on playful learning situations rather than game rewards. In fact popular microworlds such as Scratch (Resnick et al., 2009), Alice (Cooper et al., 2000) and Greenfoot (Kölling, 2009) are used to overcome difficulties in learning programming to beginners through play and metaphors (Djelil et al., 2016).

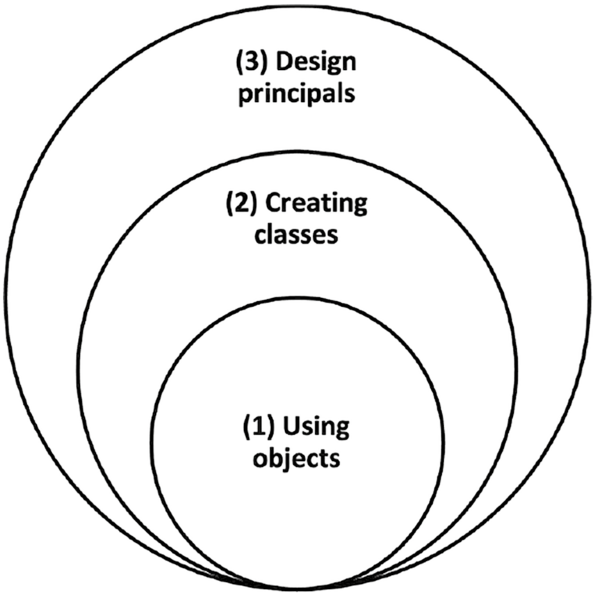


Fig. 1 Didactic engineering defining the objects-first approach

2.2.1 Ludicisation and programming microworlds

Ludicisation allows for the design of learning situations that foster playful attitude, leading students to use artefacts that are not necessarily games (Genvo, 2011). Ludicisation differs from gamification, which provides students with engaging activities through rewards and positive reinforcements (Genvo, 2012). Ludicisation is thus a way of designing playful and interactive learning situations. It is a common practice for programming learning (Seralidou & Douligeris, 2021; Combéfis et al., 2016).

Programming microworlds also implement ludicisation, e.g. Scratch (Resnick et al., 2009), Alice (Cooper et al., 2000) and Greenfoot (Kölling, 2009) aim, from one hand, to allow students to develop autonomously, through personal discovery and exploration, complex and abstract knowledge (Papert, 2020; Kafai, 2006). The learning situation induced using these environments also involves problem-solving. As a principle of the constructionism paradigm (Harel & Papert, 1991; Perkins, 2013), this allows for exploring a domain in a rich and meaningful way (Rieber, 1996). From another hand, the goal consists of providing students with engaging learning situations that allow for intrinsically motivating learning activities connected with learners' expectations through play (Papert, 2020). These environments provide meaningful experiences for learning programming, by offering students the opportunity to design and develop their own games. They allow students to create game scenarios by designing programs, leading to playful learning situations. Ludicisation is therefore an essential design principle for programming microworlds.

2.2.2 Metaphors and programming learning

Programming microworlds use metaphors to make abstract concepts more easily graspable by beginners (Xinogalos et al., 2006). In such environments, programming concepts are experienced in a graphical scene as a narrative, an animation or a significant phenomenon (Cooper et al., 2000; Kölling, 2009; Resnick et al., 2009).

In computer science, the most important computing concepts are funded on metaphors. Therefore, explicit metaphors are often used to teach beginner students how to code computer programs (McConnell, 2004). According to (Travers, 1996), computation itself is a structuring metaphor and programming models are built on metaphors. This includes the object-oriented paradigm, in which computational objects are depicted metaphorically in terms of physical and social objects (Travers, 1996): As physical objects, they have properties and state. As social objects, they communicate and interact.

Metaphors enable the formalisation and communication of new knowledge in an understandable way for learners, and provide students with analogies that foster learning (Carroll & Mack, 1999). Indeed, metaphors make programming tangible and provide students with means to understand the abstract operations of the computer in terms borrowed from more familiar domains (Travers, 1996). This is very useful for learning concepts that cannot be directly perceived. In this sense, a metaphor is defined as a way to structure and transform the knowledge from one domain through mapping concepts and relations to build another domain that is already familiar. This new domain is expected to enabling learning since the "essence of metaphor is understanding and experiencing one kind of thing in terms of another" (Lakoff & Johnson, 2008).

This transformation of knowledge consists of didactic transposition. In the following, we describe the didactic transposition framework in which programming knowledge is transformed using a metaphor in the game design.

2.3 Didactic transposition and game design

Didactic transposition is defined by a process by which knowledge (academic knowledge) is transformed to become learning objects (knowledge to teach) (Colomb, 1986; Botet, 2008). The process does not only consist of the simplification of knowledge but rather to significantly change academic knowledge to be taught and learned. Didactic transposition is a process of deconstruction and rebuilding of academic knowledge with the aim of making it teachable (Bosch & Gascón, 2006). This process includes two main steps, including how the scholarly knowledge is shaped to become knowledge to be taught, and how the teacher contextualises the knowledge to be taught into meaningful learning situations.

Bonnat et al. (2022) proposed a framework as an alternative to the classical model of didactic transposition. In this model, didactic transposition is approached from the perspective of ludicisation and metaphors. A metaphor is an implicit analogy that operates a transfer of meaning from an abstract target domain to a concrete source domain, in other words, it allows a target domain to be understood from a source domain. Thus, the didactic transposition corresponds to a conversion of a target situation into a source situation (Fig. 2). The target situation is the target abstract learning domain, and the source situation is the concrete learning situation provided to the learner.

According to this model, learning is defined as the ability for the learner to identify analogies and relationships between the source and the target situations (Hofstadter and Sander, 2013). The metaphor is a figurative meaning of the teaching domain. As a result, there is an isotopy, a common meaning, between the elements of the source situation (the game) and the elements of the target situation (the concepts to be learnt). Nevertheless, to perceive this isotopy, the learner must follow an interpretative path that will allow him to deconstruct the metaphor. This de-metaphorisation takes place during the debriefing carried out by the instructor after the game session (Bonnat et al., 2022).

3 Method

In this paper, we aim to analyse the didactic transposition by which the academic computing knowledge is contextualised in a new game-based learning environment called Progo. This analysis is conducted from two perspectives: (1) the design inherent to the objects-first approach for the introduction of object-oriented principles to beginners, and (2) the didactic transposition with the metaphorisation and ludicisation processes that operates in the game design. Our main objective is to characterise how the object-oriented concepts are transformed and reshaped in the Progo game design and the resulting learning situation. This analysis aims to evaluate the

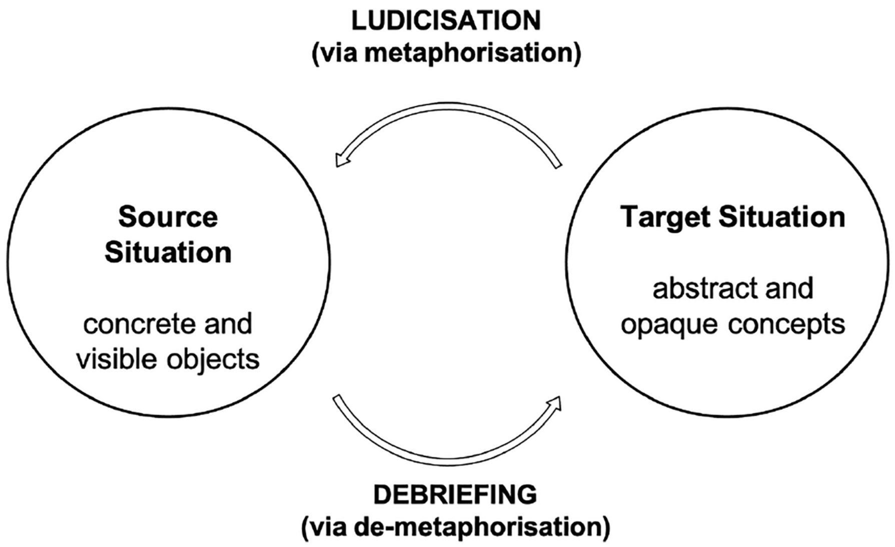


Fig. 2 Didactic transposition framework (Bonnat et al., 2022)

distance between the target situation (the knowledge to be taught) and the source situation (the game situation induced by Progo). Although several empirical studies have been conducted to analyse the use of Progo and its learning effectiveness (Djelil et al., 2015; Djelil et al., 2017; Djelil et al., 2019), we do not focus in this study on the analysis of the learning experience, but we rather aim to analyse the game design based on the didactic transposition model.

Our method relies, on the one hand, on an analysis of the curriculum in order to identify the knowledge to be taught and, on the other hand, on an *a priori* analysis of the Progo game. This *a priori* analysis (Artigue, 1988), consists of determining in what way the choices made in the design of the game (didactic transposition) are expected to impact the learners' behaviour and the learning process. The *a priori* analysis is based on the identification of the different elements of the game, the tasks they provide with and the knowledge mobilised by these tasks. This analysis aims to describe the source situation and thus to characterise the transformation of knowledge during the design of the game (RQ 1) and the analogical relations between the source situation and the target situation (RQ 2). The *a priori* analysis has close connections with conjecture mapping (Sandoval, 2014), a design method developed for design-based research. The close analysis of a learning device allows for the identification of design conjectures i.e. "conjectures about how to support learning in a specific context, that are themselves based on theoretical conjectures of how learning occurs in particular domains" (Sandoval & Bell, 2004). Design conjectures are implemented as design elements that are expected to produce specific effects on the learning process. The *a priori* analysis is performed by the authors of this article: a game-based learning scholar and a computer science education scholar.

4 Analysis of the Progo game design

4.1 Analysis of academic curricula for teaching object-oriented basics

In higher education and more particularly in the French educational system, computer science curricula include object-oriented programming from the beginning of cycle 1, and often during the core curricula (before specialisation, e.g. object-oriented design and programming courses in Universities and Engineering Schools (Roberts & et al., 2001), National Pedagogical Programme of the computer science undergraduate degree). Computing curricula generally include courses which are spread over several weeks, due to a large number of concepts and knowledge to be taught. Object-oriented programming introduction courses come often after the course "introduction to programming and algorithms". Instructors choose to use programming languages that are widespread in industries, such as Java and C++. Students therefore have some programming skills, but very often they are beginners on object-orientation and its programming languages.

According to the didactic transposition framework, the target situation consists of the object-oriented programming basics as defined in computer science curricula. We have analysed the academic curricula according to the objects-first approach, to describe the target situation. We describe the object-oriented basics with respect to the didactic engineering of the objects-first approach that embeds the fundamental concepts within the three didactic steps: 1) using objects, 2) creating classes and 3) design principles. Table 1 lists, for each didactic step, the corresponding object-oriented basics with respect to the academic curricula.

These programming concepts are the target learning concepts, i.e. the knowledge to be taught (or learning outcomes). In what follows, we analyse how these concepts are transformed and how their meaning is preserved by the analogical relationships between the target and the source situations.

4.2 Knowledge transformation in the Progo game design

The didactic transposition framework defines a target situation and a source situation: In our context, the target situation is the object-oriented programming paradigm (design and coding principles), the source situation is the Progo environment, which is based on a metaphor of construction and animation (Fig. 3).

The Progo interface comprises a three-dimensional (3D) scene, which allows the learner to design his own constructions by assembling, colouring and rotating 3D components. The interface also comprises a code editor, entirely synchronised with the 3D scene. The learner can interact with the 3D components by viewing and entering code. Therefore, the game-play consists in building and animating robots or 3D mechanical structures (Fig. 4).

The 3D components visible at the user interface and the interactions they provide allow to shape object-oriented basics through metaphorisation and ludicisation. The design of Progo embeds object-oriented concepts through three steps: 1) using objects, 2) creating classes and 3) design principles. In the following, we describe how ludicisation and

Table 1 Object-oriented fundamentals embedded within three didactic steps

Didactic step	Programming concept
1) Using objects	1.1. Relationship between an object and a class: an object results from a class instantiation.
	1.2. Object characteristics: an object is characterised by attributes and methods.
	1.3. Object state: attribute values and method calls.
2) Creating classes	2.1. Class role: a class allows for the description of object properties. It is a new data type.
	2.2. Class encapsulation: hiding internal data and methods from the outside of a class.
	2.3. Class constructor: function of a class that allows creating and initialising new objects of that class.
3) Design principles	3.1. Association: relationship between classes of objects. Each Object is connected to another, knowing its reference.
	3.2. Aggregation: symmetric association between classes representing a "ensemble/element" relationship. Both the entities can exist individually.
	3.3. Composition: form of aggregation in which two entities are highly dependent on each other, representing a "part-of" relationship. The composed object can not exist without the other entity.
	3.4. Inheritance: represent a relationship between classes, where an inherited class is a subclass of its parent class or super class. An object created through inheritance acquires all the properties of the super class.

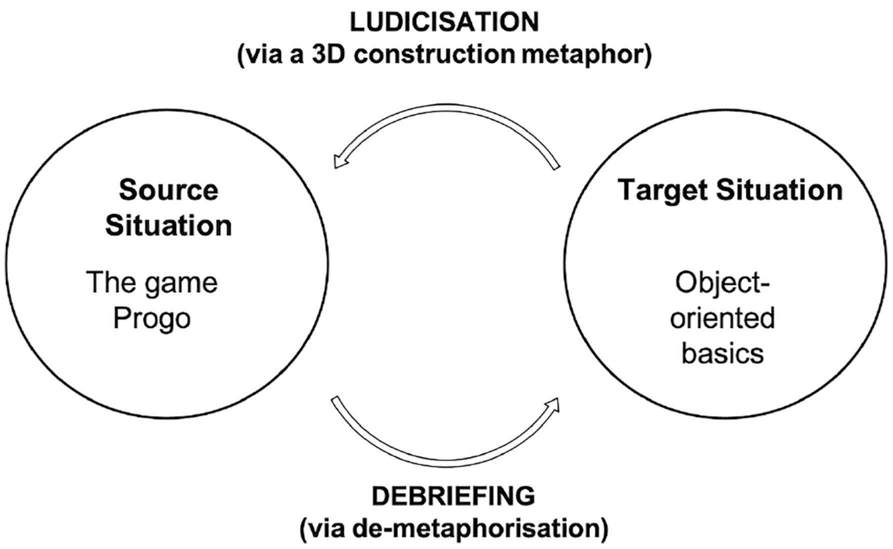


Fig. 3 Didactic transposition model applied to the Progo game

metaphorisation reshape the object-oriented concepts in a meaningful and playful way, with regard to the three didactic steps (RQ 1). We also describe the analogical relationships between the source situation and the target situation (RQ 2). We first consider the

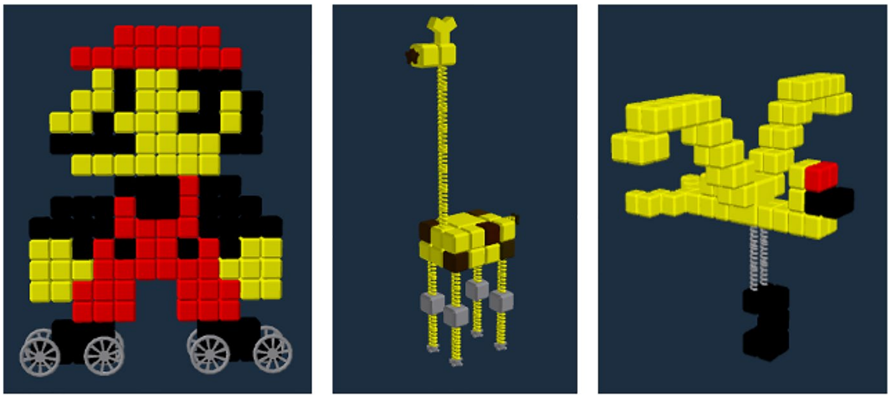


Fig. 4 3D constructions realised by students when playing with Progo

analogy between using existing 3D components in the Progo game with the didactic step using objects. Second, we consider the analogy between the creation of a new 3D construction and the didactic step creating classes. Finally, we consider the analogy between 3D constructions with the didactic step using design principles.

4.2.1 Using existing 3D components

During the first didactic step, the learner starts with manipulating objects. As a result, he experiences the following concepts: 1) The relationship between an object and its class, 2) The setting object attributes, 3) The modification of attribute values and method calls (Table 1).

The Progo interface makes visible the concept of a class through a 3D graphical model. Each time a model is instantiated, an interactive object is created. When an object is created, its visual appearance is similar to its class.

The characteristics of an object are expressed through its appearance and behaviour. The appearance includes the object position (its location relatively to another object), its colour and its rotation angle. These characteristics are the object attributes. The behaviour of an object includes the ability to be assembled with another object to build a more complex structure, the ability to change its colour and/or the ability to rotate for a given duration. These characteristics are object methods.

The learner can change an object appearance by modifying the values of its attributes or making method calls. This is what defines an object state.

The learner experiences these concepts both with the 3D scene and the code editor (Fig. 5). Once the learner has finished his construction, he is invited to create a new class with his realisation in a second phase.

4.2.2 Creating a new 3D construction

During the second didactic step, the learner moves to the concept of class, comprising: 1) the class role, 2) the class encapsulation, and 3) the class constructor (Table 1).

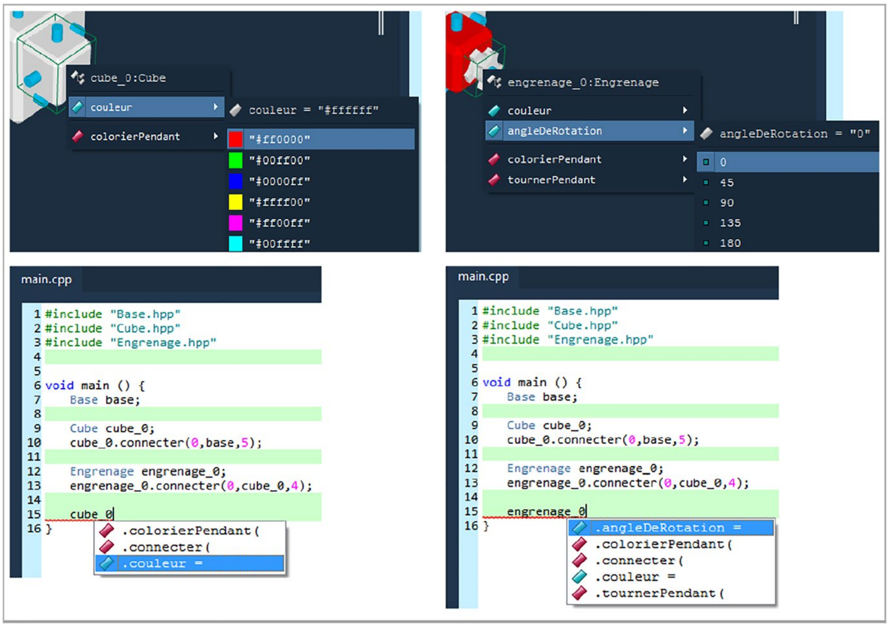
With the Progo interface, the learner can name and save each new 3D construction as a new class which can be instantiated and visualised in the 3D scene. This new graphical model gathers all the properties and behaviours of the objects chosen beforehand by the learner during the previous didactic step. Therefore, the player experiences the class role.

Once a new class is created, the learner can visualise new tabs displaying two codes in C++ programming language, the declaration of the new class ("*.hpp" file), and the definition of this class ("*.cpp" file) in the code editor. The learner can therefore experience the concept of encapsulation by observing the codes and by trying to manipulate new instances of this new class in a new "main()" function. The learner may notice that some of the attributes and methods of the new class are "private" (preceded by the keyword "private") and others are "public" (preceded by the keyword "public"). The learner also has the opportunity to make calls to the public methods in the "main()" function as well as in the 3D scene, but not in the private ones.

The new class also has a constructor which allows an object to be initialised as soon it is created. When observing the constructor code, the learner may notice that the constructor is preceded by the keyword "public", has the same name as its class, and does not have a return type (Fig. 6).

4.2.3 3D constructions as object-oriented systems

During the third didactic step, the learner is introduced to the design principles of the object-oriented paradigm. In the academic curricula, this mainly comprises the principles of association, aggregation, composition and inheritance (Table 1).



```
#include "Base.hpp"
#include "Cube.hpp"
#include "Engrenage.hpp"


void main () {
    Base base;

    Cube cube_0;
    cube_0.connecter(0,base,5);

    Engrenage engrenage_0;
    engrenage_0.connecter(0,cube_0,4);

    cube_0
}
```

```
#include "Base.hpp"
#include "Cube.hpp"
#include "Engrenage.hpp"


void main () {
    Base base;

    Cube cube_0;
    cube_0.connecter(0,base,5);

    Engrenage engrenage_0;
    engrenage_0.connecter(0,cube_0,4);

    engrenage_0
}
```

Fig. 5 Modifying attribute values and making method calls, both in the 3D scene and in the code editor

In the Progo environment, the building of each 3D construction uses classes of objects which are connected to each others. Therefore, relationships between classes of objects manipulated by students when playing, are consistent with the object-oriented design principles. In this didactic step, the instructor can ask students to analyse the design principles that are illustrated in the 3D constructions.

Regarding the object-oriented paradigm, each 3D construction is an object-oriented system, that can be modelled using the Unified Modeling Language (UML) (Muller & Gaertner, 2000), which is a very widespread formalism used in the french academic curricula to introduce object-oriented design principles. UML allows to describe in a formal way the object-oriented relationships between the different classes used to build a 3D construction in Progo (Fig. 7).

The classes of objects in the Progo construction game are modelled as follows:

1. The "3DConstruction" class: This class is a model of the final realisation of a student. It is a composition (part-of relationship) of all the classes used by the students during the game. It defines two main methods: the method "animate()" that gathers all the animation operations programmed by the student, and the method "reinitialise()", that allows to set a 3D construction to its initial state.
2. Abstract classes that are not visible at the interface, but allow to model the visible ones as graphical components having some characteristics. They are modelled as follows:

    - The "3DComponent" class: models all the graphical components, having the attribute "color", and the methods "connect()", and "colorForaDuration()", allowing respectively to connect an object to another, and to colour an object during a period of time (animation effect).
    - The class "ActiveComponent": This class models all the graphical components, having additional characteristics comparatively with the precedent

```
class Robot {
private : // private members (encapsulation)
      Base base;
      BifurcationY bifurcation_0;
      Cube cube_0;
      Engrenage engrenage_0;

void initialiserAngle();
void initialiserCouleur();

public : //public methods
     Robot(); //the Robot class constructor
     Void animer();
     Void reinitialiser (bool angle, bool couleur);
};
```

Fig. 6 Generated declaration code of a new class called Robot

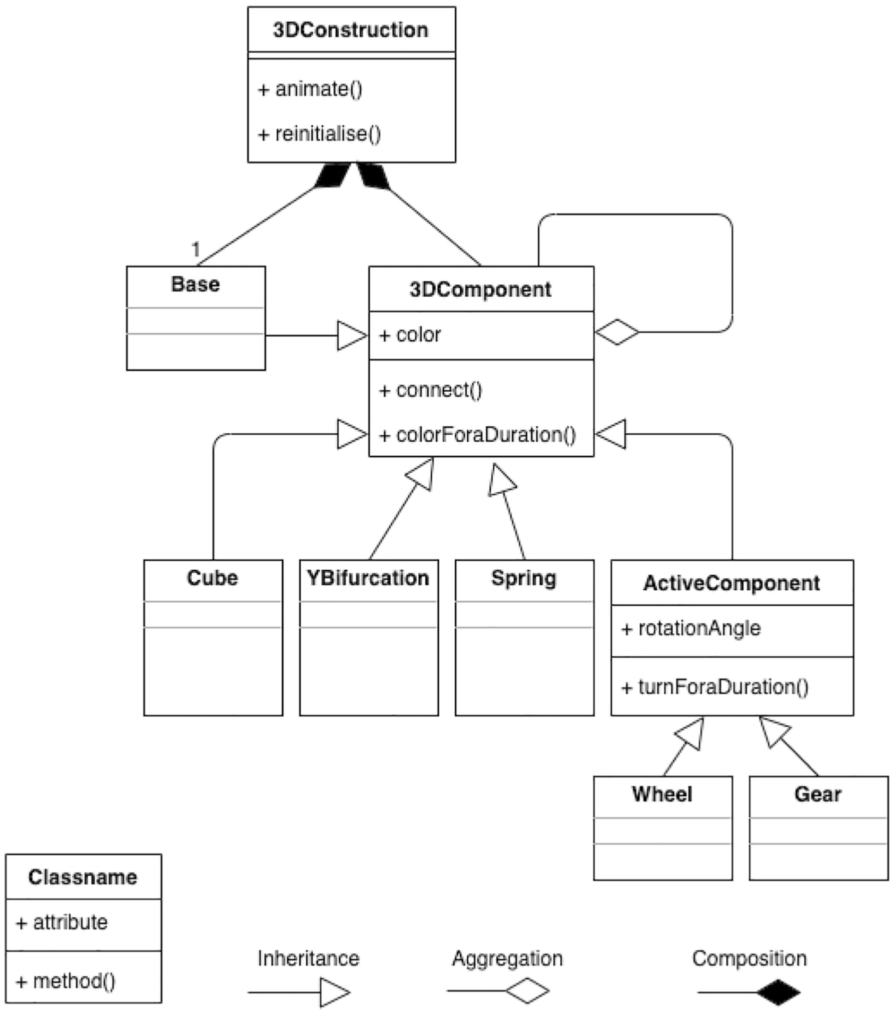


Fig. 7 UML Class diagram modelling a Progo 3D construction

class. They are called active components since they are able to perform a movement as a result of a rotation during a period of time. This leads to animation effect. This class is inherited from the ”3DComponent” class, and defines the new attribute “rotationAngle”, and the method “turnFora-Duration()”.

3. Visible classes at the interface are classes that the students can directly manipulate when playing (Fig 8). They are inherited from the abstract classes, either from the class “3DComponent” such as “Base”, “Cube”, “YBifurcation” and “Spring”, or from the class “ActiveComponent”, such as “Wheel” and “Gear”. The class “Base” has a unique instance as a construction basis. It is visible in the 3D scene when launching the Progo environment, inviting students to start playing.

Finally, in addition to these different classes, the Progo construction game enables the introduction of three distinct design principles from the academic curricula (see Table 1) :

1. Inheritance: relationships between each of the classes,
    - "Base", "Cube", "YBifurcation", "Spring" and "ActiveComponent" with the class "3DComponent".
    - "Wheel" and "Gear", with the class "ActiveComponent".
2. Aggregation: the ensemble/element relationship between all the instances of the class "3DComponents", since each component is connected to each other, but removing some components doesn't impact others.
3. Composition: relationship between the "3DConstruction" class with the "3DComponent" class from one hand, and the "Base" class from another hand. The composed 3D construction cannot exist without the composing entities.

5 Discussion

The concept of ludicisation, as a form of didactic transposition (Balacheff, 1994; Bonnat et al., 2022), allows to distinguish between a target situation (in the sense of a learning target), which integrates the knowledge to be taught, and a source situation (in the sense of a learning source), that refers to the learning situation in which this knowledge is contextualised in the form of playful problems to be solved, tasks to be carried out or even objects to be manipulated.

In our work, knowledge is integrated into a learning situation through a metaphor of a construction game dedicated to the creation and animation of 3D objects. This metaphor transforms the characteristics of object-oriented programming concepts (target situation), into graphical 3D objects to be manipulated in a construction game (source situation).

The process of didactic transposition has captured the essence of the target situation similarly to a literary metaphor. Indeed, the metaphor is a refined form of the target situation, aiming to allow learners focusing their attention on the core concepts of this situation. In our case, the core situation comprises key concepts of object-oriented programming (e.g. the concepts of objects, class and relationship

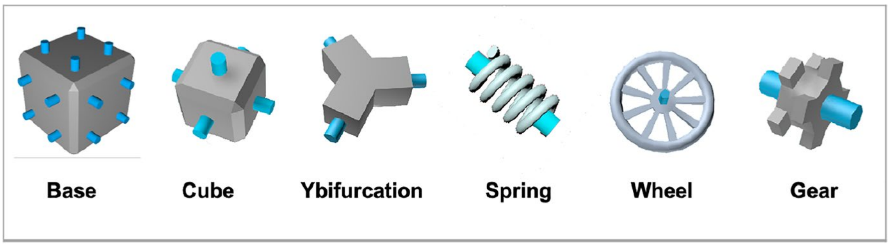


Fig. 8 3D graphical components representing classes of objects

between the object and class are represented by visible and interactive 3D graphics, which are instances of 3D graphical models, setting attribute values and making method calls are illustrated by colouring, rotating or connecting graphics).

The construction metaphor embedded in the game is a well known and intuitive conceptual domain (the source situation), which is projected on the conceptual domain to be learned (the target situation). This allows the learner to move from concrete and visible concepts (3D graphics and their characteristics) to more abstract and opaque concepts (objects, classes, methods, etc.) (Botet, 2008). In other words, the learner moves from the experience of the construction game to the computing concepts to be learned.

There is an analogical relationship between the situation implied by the Progo environment (source situation) and the object-oriented concepts (target situation). Indeed, the construction metaphor embedded in the Progo game allows the description of object-oriented basics in a reliable and accurate way (objects, classes and design principles). This is in line with (Hofstadter & Sander, 2013), who assumes that learning does not result from a purely interpretative logic, but rather from a process that leads the learner to identify analogical relationships between different situations.

Therefore, the analysis we have carried out allow us to answer the research questions (RQ 1 and RQ 2). Contextualisation of computing knowledge in the game Progo is a result of a transformation process, that maintained an analogical relationship between the knowledge concerned and the game metaphor.

6 Limitations

This work highlights how didactic transposition operates on the design of the Progo environment through metaphorisation and ludicisation. However, our analysis doesn't take into consideration the reverse process, which consists of de-metaphorisation. In practice, this takes the form of a debriefing conducted by the teacher, also referred to as knowledge institutionalisation (Brousseau & Balacheff, 1998). A student who plays with Progo develops knowledge which needs to become communicable and transferable to other situations. Indeed, the French language recognises a difference between "situational knowledge" (*connaissance*) and "institutional knowledge" (*savoir*) (Plumettaz-Sieber et al., 2019). A student who plays Progo develops knowledge which is implicit, subjective and situated. For example, the concept of *object* is not explicit within the Progo environment. This concept is only reflected through the tasks performed by the player. It means that, although the player learns how to use *an object* by playing Progo, he is not able to explain what *an object* is, in computing terms. Warfield (2013) uses the expression "to be familiar with" to describe the knowledge gained by students involved in an autonomous activity such as Progo. This situational knowledge is not formalised and tied to a specific context (i.e, the learning context). The transfer of this knowledge implies a transformation called institutionalisation (Brousseau, 1997), and this occurs during debriefing (Plumettaz-Sieber et al., 2019). Institutionalisation, relates to the change of the status of knowledge. The implicit, subjective and situated knowledge becomes explicit, objective and context-free (institutional knowledge). (Warfield, 2013) uses the expression "to know a fact" to stress that institutionalisation consists of a metacognitive process. This transformation of knowledge is under the responsibility of the teacher. During debriefing the game is over, the players become learners, and, through questions and discussions, the concepts experienced when playing the game are named, explained and validated. In addition, future usages of the knowledge are considered. Thus, institutionalisation of knowledge and debriefing are the counter parts of metaphorisation and ludicisation. While metaphorisation and ludicisation occur on the design of the game, institutionalisation occurs during the debriefing which follows the time dedicated to play the game. However, both processes are essential.

7 Conclusions and implications

In this paper we performed an *a priori* analysis of the game Progo, a new environment dedicated to learning object-oriented programming basics to beginners. Our analysis relies on the didactic transposition framework, which allows, on the one hand, to highlight how computing knowledge is transformed and reshaped to be "played" as more intuitive and graspable objects, and from another hand, to verify whether analogies between the computing knowledge and the manipulated objects are maintained during this transformation process. The didactic transposition framework allows to distinguish between a target situation (target knowledge, abstract and opaque concepts, i.e. the object-oriented programming basics) and a source situation (concrete and visible objects, i.e. the game play induced by Progo), and highlights a transformation process between these two situations as a result of metaphorisation and ludicisation process (i.e. the Progo construction and animation game metaphor).

Moreover, in order to be able to describe the target situation in a comprehensible way, we based our analysis on the objects-first approach. This approach embeds object-oriented concepts in three didactic steps that help beginners to progressively master the targeted concepts. This approach is part of the Progo game design and provides a means for the description of the programming concepts as defined in computer science curricula. An *a priori* analysis of the game, allowed to describe in an effective way, how each programming concept is transformed in the game, through a construction and animation metaphor, in terms of 3D objects and learner's interactions fostered by this transformation (e.g. a visible and interactive 3D component is a metaphor of a computing object, colouring or rotating a 3D component is a metaphor of a method call, ...). The choices made through this metaphor, aim to trigger a play situation when learners interact with the construction game (i.e. the source situation). This analysis showed clear analogies between the source and target situations, and the meaning of the considered concepts is maintained.

This work is a contribution to computer science didactic and game design. Indeed, didactic transposition approached from the perspective of metaphorisation and ludicisation, revealed a powerful tool for game design analysis in the case of object-oriented programming learning. In addition, debriefing was identified as an essential element in the didactic transposition process. Indeed, debriefing is the reverse process of metaphorisation. It provides students with a means to take a step backward after the game session and to identify the knowledge embedded in the metaphor, and, by doing so, to be able to transfer the gained knowledge to other situations. Indeed, if this paper describes the process of contextualising knowledge in a metaphorical and playful learning situation, this process needs to be reversed by another process that allows the conversion of subjective and implicit knowledge into explicit knowledge. Both processes consist of important and needed transformations of knowledge that educators should take into consideration.

This work open new perspectives for understanding how game-based learning occurs. In particular, it motivate future work about the role of debriefing to helping students to understand the analogical relationships between the source and the target situation. For example, it could be relevant to study whether students establish links between the object-oriented programming concepts and the properties of the graphical objects they see and manipulate with the Progo interface, and how this can be enhanced by debriefing. It would be also interesting to analyse to what extent students are able to transfer what they learn to other similar situations. Indeed, we consider that game-based learning does not only results from gaming. Game-based learning results from a metacognitive process on the gaming experience which allows for the transfer of knowledge.

References

Abbasi, S., Kazi, H., & Khowaja, K. (2017). A systematic review of learning object oriented programming through serious games and programming approaches. In *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS), (pp. 1–6).*

Abidin, Z. Z., & Zawawi, M. A. A. (2020). Oop-ar: Learn object oriented programming using augmented reality. *International Journal of Multimedia and Recent Innovation*, *2*(1), 60–75.

Adams, J., & Frens, J. (2003). Object centered design for java: teaching ood in cs-1. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education, (pp. 273–277).*

Artigue, M. (1988). Ingénierie didactique. *Recherches en didactique des mathématiques*, *9*(3), 281–308.

Balacheff, N. (1994). La transposition informatique. note sur un nouveau problème pour la didactique. *Vingt ans de didactique des mathématiques en France*, *2*, 132–138.

Becker, B. W. (2001). Teaching cs1 with karel the robot in java. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, (pp. 50–54).*

Bennedsen, J. (2008). Teaching and learning introductory programming: a model-based approach.

Bennedsen, J., & Schulte, C. (2007). What does" objects-first" mean? an international study of teachers' perceptions of objects-first. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88, (pp. 21–29).*

Bersini, H. (2017). La programmation orientée objet. Editions Eyrolles.

Bonnat, C., Sanchez, E., Paukovics, E., & Kramar, N. (2022). Didactic transposition and learning game design. proposal of a model integrating ludicization, and test in a school visit context in a museum. In *Didactics in a Changing World. European Perspectives on Teaching, Learning and the Curriculum: EERA Book Series.*

Bosch, M., & Gascón, J (2006). Twenty-five years of the didactic transposition. *ICMI bulletin*, *58*(58), 51–65.

Botet, S. (2008). Petit traité de la métaphore, un panorama des théories modernes de la métaphore. Presses universitaires de Strasbourg.

Brousseau, G. (1997). Theory of didactical situations in mathematics (n. balacheff, m. cooper, r. sutherland & v. warfield: Eds. and trans). Dordrecht: Kluwer.

Brousseau, G. (2006). Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990 (Vol. 19). Springer Science & Business Media.

Brousseau, G., & Balacheff, N. (1998). Théorie des situations didactiques: Didactique des mathématiques 1970-1990. La pensée sauvage Grenoble.

Buck, D., & Stucki, D. J. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. *ACM SIGCSE Bulletin*, *32*(1), 75–79.

Carroll, J. M., & Mack, R. L. (1999). Metaphor, computing systems, and active learning. *International Journal of Human-Computer Studies*, *51*(2), 385–403.

Colomb, J. (1986). Chevallard (yves).la transposition didactique: du savoir savant au savoir enseigné. *Revue française de pédagogie*, *76*(1), 89–91.

Combéfis, S, Beresnevičius, G, & Dagienė, V (2016). Learning programming through games and contests: overview, characterisation and discussion. *Olympiads in Informatics*, *10*(1), 39–60.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-d tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15* (5), 107–116.

Costa, J. M., & Miranda, G. L. (2017). Relation between alice software and programming learning: A systematic review of the literature and meta-analysis. *British Journal of Educational Technology*, *48*(6), 1464–1474.

Dabancourt, C. (2008). Apprendre à programmer: algorithmes et conception objet. Editions Eyrolles.

Djelil, F., Albouy-Kissi, A., Albouy-Kissi, B., Sanchez, E., & Lavest, J-M. (2016). Microworlds for learning object-oriented programming: Considerations from research to practice. Journal of Interactive Learning Research, 27(3).

Djelil, F., Albouy-Kissi, B., Albouy-Kissi, A., Sanchez, E., & Lavest, J.-M. (2015). Towards a 3d virtual game for learning object-oriented programming fundamentals and c++ language theoretical considerations and empirical results. International Conference on Computer Supported Education.

Djelil, F., Montesinos, M. T. S., & Gilliot, J.-M. (2020). Une approche didactique pour l'introduction de la programmation orientée-objet en classe. DIDAPRO-8.

Djelil, F., Muller, P.-A., & Sanchez, E. (2019). Investigating learners' behaviours when interacting with a programming microworld. In D Passey, R Bottino, C Lewin, & E Sanchez (Eds.) *Empowering Learners for Life in the Digital Age* (pp. 67–76). Cham: Springer International Publishing.

Djelil, F., Sanchez, E., Albouy-Kissi, B., & Albouy-Kissi, A. (2017). Acquisition de connaissances de programmation en fonction des stratégies d'apprentissage: une étude empirique du micromonde progo. EIAH 2017, (pp. 41–52).

Fabricatore, C. (2000). Learning and videogames: An unexploited synergy.

Genvo, S. (2011). Penser les phénomènes de ludicisation du numérique: pour une théorie de la jouabilité. Revue des sciences sociales (Strasbourg) (45).

Genvo, S. (2012). La théorie de la ludicisation: une approche anti-essentialiste des phénomènes ludiques. Journée d'études Jeu et jouabilité à l'ère numérique.

Habgood, M. P. J. (2007). The effective integration of digital games and learning content (Unpublished doctoral dissertation). University of Nottingham Nottingham.

Harel, I. E., & Papert, S. E. (1991). Constructionism. Ablex Publishing.

Hazzan, O., Dubinsky, Y., & Meerbaum-Salant, O. (2010). Didactic transposition in computer science education. *ACM Inroads*, *1*(4), 33–37.

Hofstadter, D. R., & Sander, E. (2013). Surfaces and essences: Analogy as the fuel and fire of thinking. Basic books.

Kafai, Y. B. (2006). Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and culture*, *1*(1), 36–40.

Ke, F. (2016). Designing and integrating purposeful learning in game play: A systematic review. *Educational Technology Research and Development*, *64*(2), 219–244.

Keung, J., Xiao, Y., Mi, Q., & Lee, V. C. (2018). Bluej-uml: Learning object-oriented programming paradigm using interactive programming environment. In *2018 International Symposium on Educational Technology (ISET), (pp. 47–51).*

Kölling, M. (2009). Introduction to programming with greenfoot. Pearson Education, Upper Saddle River, New Jersey, USA.

Kolling, M. (2015). Introduction to programming with greenfoot: Object-oriented programming in java with games and simulations. Pearson.

Kölling, M, & Rosenberg, J. (2001). Guidelines for teaching object orientation with java. *ACM SIGCSE Bulletin*, *33*(3), 33–36.

Krugel, J., & Hubwieser, P. (2018). Strictly objects first: A multipurpose course on computational thinking. In *Computational Thinking in the STEM Disciplines, (pp. 73–98). Springer.*

Lakoff, G., & Johnson, M. (2008). Metaphors we live by. University of Chicago press.

McConnell, S. (2004). Code complete. Pearson Education.

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, *62*(2), 77–90.

Michaelson, G. (2018). Microworlds, objects first, computational thinking and programming. In *Computational Thinking in the STEM Disciplines, (pp. 31–48). Springer*.

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, (pp. 75–79).*

Muller, P-A, & Gaertner, N. (2000). Modélisation objet avec uml. (Vol 514) Eyrolles Paris.

Orange, C. (1990). Didactique de l'informatique et pratiques sociales de référence. Bulletin de l'EPI (Enseignement Public et Informatique). (60) 151–161.

Papert, S. A. (2020). Mindstorms: Children, computers, and powerful ideas. Basic books.

Perkins, D. N. (2013). Knowledge as design. Routledge.

Piteira, M., & Costa, C. (2013). Learning computer programming: study of difficulties in learning programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication, (pp. 75–80).*

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of game-based learning. *Educational Psychologist*, *50*(4), 258–283.

Plumettaz-Sieber, M., Bonnat, C., & Sanchez, E. (2019). Debriefing and knowledge processing an empirical study about game-based learning for computer education. In *International Conference on Games and Learning Alliance, (pp. 32–41).*

Resnick, M., Maloney, J., Monroy-Hernández, A, Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & et al. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Rieber, L. P. (1996). Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational Technology Research and Development*, *44*(2), 43–58.

Roberts, E., et al. (2001). Computing curricula 2001.

Sanchez, E. (2019). Game-based learning. In A Tatnall (Ed.) *Encyclopedia of Education and Information Technologies*. https://doi.org/10.1007/978-3-319-60013-0_39-2 (pp. 1–9). Cham: Springer International Publishing.

Sandoval, W. (2014). Conjecture mapping: An approach to systematic educational design research. *Journal of the Learning Sciences*, *23*(1), 18–36.

Sandoval, W. A., & Bell, P. (2004). Design-based research methods for studying learning in context: Introduction. *Educational Psychologist*, *39*(4), 199–201.

Seng, W. Y., & Yatim, M. H. M. (2014). Computer game as learning and teaching tool for object oriented programming in higher education institution. *Procedia-Social and Behavioral Sciences*, *123*, 215–224.

Seralidou, E., & Douligeris, C. (2021). Learning programming by creating games through the use of structured activities in secondary education in greece. *Education and Information Technologies*, *26*(1), 859–898.

Travers, M. D. (1996). Programming with agents new metaphors for thinking about computation (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Warfield, V. M. (2013). Invitation to didactique (Vol 30). Springer Science & Business Media.

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in k-12 school curricula of the 2lst century: Why, what and when?. *Education and Information Technologies*, *22*(2), 445–468.

Woei, L. S., Othman, I. H., & Man, C. K. (2015). Learning programming using objects-first approach through folktales. Jurnal Teknologi 75(3).

Woodworth, P., & Dann, W. (1999). Integrating console and event-driven models in cs1. *ACM SIGCSE Bulletin*, *31*(1), 132–135.

Xinogalos, S., Satratzemi, M., & Dagdilelis, V. (2006). An introduction to object-oriented programming with a didactic microworld: objectkarel. *Computers & Education*, *47*(2), 148–171.

Yukselturk, E., & Altiok, S. (2017). An investigation of the effects of programming with scratch on the preservice it teachers self-efficacy perceptions and attitudes towards computer programming. *British Journal of Educational Technology*, *48* (3), 789–801.