Review of the Adaptability of a Set of Learning Games Meant for Teaching Computational Thinking or Programming in France

Hajar Saddoug1, Aryan Rahimian1, Bertrand Marne2, Mathieu Muratet3, Karim Sehaba4 and Sébastien Jolivet5

1Sorbonne Université, Place Jussieu, Paris, France
2ICAR UMR 5191, Université Lumière Lyon 2, Parvis René Descartes, Lyon, France
3Sorbonne Université, CNRS, INS HEA, LIP6, F-75005 Paris, France
4LIRIS - Université Lumière Lyon 2, avenue Pierre Mendès-France, Lyon, France
5IUEF, Université de Genève, rue du Général-Dufour, Genève, Switzerland

hajar.saddoug@etu.sorbonne-universite.fr, rahimian_aryan@yahoo.fr, bertrand.marne@ens-lyon.fr, Mathieu.Muratet@lip6.fr, karim.sehaba@liris.cnrs.fr, Sebastien.jolivet@unige.ch

Keywords: Serious Games, Meta-design, Instrumental Genesis, Computational Thinking, Programming.

Abstract: Our work is part of a broader research project on how French teachers and trainers can appropriate learning games dedicated to computer science and programming. To foster this appropriation, we aim to implement a meta-design approach that favours instrumental genesis. In this article, we describe our review of a selection of learning games to identify whether they are suitable for this approach. We introduce a set of rather generic and reusable criteria to characterize the instrumentalization of serious games. With these criteria we thoroughly review 10 games among 48 selected. Thus, for each game selected, the identified criteria point out the availability of means and tools allowing teachers and trainers to understand the game, but they also assess adaptability of the latter in relation to pedagogical needs. Our results show that the adaptability of most of these 10 games remains weak and out of reach for many teachers and trainers. Indeed, none of the reviewed games were able to meet the requirements set out in the framework of the meta-design approach.

1 INTRODUCTION

In our research context, education in France, the question of computer science education dates back to the 1970s and is characterized by a balance between two conceptions of what should be taught. On the one hand, there is the idea of teaching IT and computer science as a tool. On the other hand, the idea of computer science as an academic subject, with its own concepts and methods to be taught (Baron & Drot-Delange, 2016). After its demise years ago, computer science has made a comeback in French school curricula, where it is referred to as “computational thinking”. According to Jeannette Wing, computational thinking involves five cognitive abilities: (1) algorithmic thinking, (2) abstraction, (3) evaluation, (4) decomposition, and (5) generalization (Wing, 2006). The French researcher Gilles Dowek (Dowek, 2011) has structured computer science into four concepts so that the content taught provides an accurate picture of the discipline itself: (1) digital information, (2) algorithms, (3) languages, and (4) computing machines. Thus, computer science is not reduced to coding, which constitutes only the final phase of the translation into a programming language. Conceived in such a fashion, computer science leaves the status of a tool for what it really is: a science with its own specificities and requiring it’s a proper learning process. However, teaching computational thinking in France at both elementary and high school levels requires a major involvement from teachers (Kradolfer et al., 2014) in order to tackle this new discipline, due to a lack of training (both pre-service and in-service).
Alongside these changes in computer science teaching, serious learning games have emerged and offer many advantages over traditional teaching tools. Indeed, many authors consider serious learning games as promising, especially for increasing learners’ engagement and motivation (Bouvier et al., 2014; Garris et al., 2002; Keller, 1995; Malone & Lepper, 2005; Prensky, 2004), while others consider these tools to foster a more constructivist learning (Bogost, 2007, 2013; Brunet et al., 2020; Gee, 2009; Ryan et al., 2012). Regarding teaching computer science and programming, many serious games exist (Miljanovic & Bradbury, 2018; Vahldick et al., 2014), but their appropriation by teachers remains scarce. To tackle this issue, we make the hypothesis that a participatory design method such as meta-design could be suitable because it is promoting instrumental genesis (Rabadel, 1995, 2003). Meta-design is an advanced participatory design method, in which the end users (“owners of problems”), in our case the teachers, are intimately involved in the initial design phases. But, and that is the reason for meta-design relevance in solving appropriation problems, users must also have the means to continue to act as designers during the use phases of the artefacts. It is possible, thanks to underdesign, which Fischer et al. (Fischer et al., 2004) define as: “[…] underdesign aims to provide social and technical instruments for the owners of problems to create the solutions [of their problems] themselves at use time”. Therefore, in our case, the goal is to identify how teachers and trainers, who are the end users of serious games dedicated to programming, manage to take part in a meta-design approach through the instrumentalization of the artefact.

Nevertheless, in order to implement this meta-design approach, it is necessary to have flexible artefacts (in our case, serious games) providing functionalities allowing these end users (teachers or trainers) to monitor their use at different levels of abstraction and to adapt them accordingly, taking into account their context and needs. In this article, we carry out a review of learning games on computer thinking with a focus on the tools made available to promote the instrumental genesis and appropriation of these games by teachers. The goal is to assess adaptability of each of the games reviewed through a meta-design approach.

In the first part, we present our methodology, i.e. how we developed our assessment criteria, how we constructed our selection of games to be assessed and how we conducted the review. In the second part, we present and discuss the results of this review, before concluding in the last part.

2 METHODOLOGY

There are already several reviews of serious games for programming in the literature. Among the research that we have been able to consult, we have noticed that the focus is mainly on the content and the skills taught by the serious games reviewed (Lindberg et al., 2019; Malliarakis et al., 2014; Miljanovic & Bradbury, 2018; Vahldick et al., 2014). According to some of these studies, such as Lindberg et al. (Lindberg et al., 2019), there is a poor alignment between the skills taught in serious games (29 were reviewed) and those taught in the curricula (particularly in French). This tendency is also found in Malliarakis et al. and Miljanovic and Bradbury (Malliarakis et al., 2014; Miljanovic & Bradbury, 2018), with respectively 12 and 49 games assessed (and many overlaps between these studies). However, these works never identified the possibility of an instrumental genesis, for instance through instrumentalization. Only Vahldick et al. (Vahldick et al., 2014) discuss the importance that editors tools can play in specific cases, yet without making it a criterion for their review (40 games reviewed).

Therefore, we conducted our own review by focusing on the ability of appraised games to have, on the one hand, monitoring capabilities that allow the teacher to understand how the game is used and, on the other hand, adaptation capabilities that allow the teacher to modify the game according to the results of the monitoring.

2.1 Learning Games Selection

To compile a list of games related to computational thinking for review, we combined the games we had already identified in other research (Marne, Muratet, et al., 2021; Marne, Sehaba, et al., 2021), with those from the above-mentioned systematic reviews, and with others searched on the web. We only retained serious games that are currently functional (for example, games using Flash technology are now very difficult to use), affordable or free. As a result, we have been able to identify 48 games that can be used to learn to code or computational thinking (Table 1).

We focus on serious games intended for learning computer science and programming in a school context, so we chose to exclude games with no real access to programming by the player, as well as those only available on mobile phones or tablets (restricted in French schools). We also merged the games with similar features to Scratch (Resnick et al., 2009) or Blockly (Fraser, 2015), both of which are microworlds...
Table 1: Initial list of 48 games, the 10 selected games are highlighted.

<table>
<thead>
<tr>
<th>Algoblocs</th>
<th>Code hunt</th>
<th>CodeWars</th>
<th>Elevator Saga</th>
<th>Hocus Focus</th>
<th>Pyrates</th>
<th>SQL Murder Mystery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bee Bot</td>
<td>Code Monkey</td>
<td>Collabots</td>
<td>Flexbox Defense</td>
<td>Imagi</td>
<td>Ruby Warrior</td>
<td>Tynker</td>
</tr>
<tr>
<td>Blockly</td>
<td>Code Monster</td>
<td>Compute It</td>
<td>Flexbox Froggy</td>
<td>KoduGame</td>
<td>Run Marco</td>
<td>Unruly Splits</td>
</tr>
<tr>
<td>Ceebot</td>
<td>CodeCombat</td>
<td>CSS Diner</td>
<td>Gladiabots</td>
<td>Le chevalier de la programmation</td>
<td>Scratch</td>
<td>Untrusted</td>
</tr>
<tr>
<td>CheckIO</td>
<td>Codefi</td>
<td>Cyber Dojo</td>
<td>Grid Garden</td>
<td>Pixel</td>
<td>Screeps</td>
<td>Vim adventures</td>
</tr>
<tr>
<td>Code</td>
<td>CodeGym</td>
<td>Duskers</td>
<td>Heartbreak</td>
<td>ProgAndPlay</td>
<td>SPY</td>
<td></td>
</tr>
</tbody>
</table>

(Papert, 1987) offering programming with blocks of instructions.

Then, we filtered the list of games, while sorting and categorizing the games by considering the following characteristics:

- Similarities and differences between the games;
- Advantages and disadvantages of the tools available within or with the games;
- Categories (e.g. age, field, cost);
- Type of training (when provided by the game): self-training, traditional training, etc.

This classification allowed us to refine the list to focus on a selection of 10 games (out of 48) that have little in common and that corresponds well to our objective: to identify serious games focused on learning computer thinking and programming that could be adapted or easily appropriated by teachers and trainers, i.e. allowing the implementation of a meta-design context. Henceforth, each game selected for review is associated with a separate category.

2.2 Preparation of the Assessment Framework

In order to establish our assessment framework, we identified several criteria. First, we identified three stages related to the possible adaptations that could be made by teachers or trainers to the listed games. For each of these three stages (before, during and after), we identified, as a criterion, specific types of support to adaptation:

- **Before Adaptation**: availability of tutorials and explanations in different formats, regarding how to handle the game and/or how to use it for other training purposes.
- **During Adaptation**: ability to check for errors during the game modification.
- **After Adaptation**: availability of an overview of the modifications made to the game and/or the presence of additional help (human or interactive digital maintenance such as forums, FAQs, etc.)

Furthermore, as a follow-up to this preparatory step, we defined more refined and easily measurable criteria. These criteria are detailed in the next section.

2.3 Refined Criteria for the Review

The purpose of this study is not to highlight some of the serious games and disqualify others. The purpose is to review the serious games, to sort them out with fine-grained criteria, in order to provide a clear framework covering both convergences and divergences among the adaptable serious games dedicated to the teaching of programming or computational thinking.

Given its subjectivity, we were reluctant to use a score system, especially as the main point of our framework was to indicate the actual availability of the defined criterion. Accordingly, we referred to the criterion as “available” or “not available”, and in the case of availability, we always comment on how the criterion is provided. However, for one of the criteria, the “Extent” of the scenario, we had to be able to be more precise, without going into too much detail for a quick analysis. Therefore, we settled on 3 possible states: small, wide or no scenario.

Once criteria were identified during our explorations, we decided to classify them. We identified 7 classes, each with different criteria, which are overviewed in Table 2.
Adaptability is crucial to our framework, describing the ability of the game to be modified. It is broken down into 4 criteria:

- **Open Source Code**: we checked the availability of the source code, its licence, and the availability of any digital resources that have been used.
- **Teacher Profile**: we checked whether a specific account and profile exist for trainers or teachers, giving access to specific features (creating lessons, planning them, viewing logs, scores, etc.).
- **HMI Modification**: we checked whether the game's interface could be modified.
- **Interaction Types**: we checked whether it was possible to modify the existing interactions in the game (e.g. using other devices than the keyboard and mouse).

The Editing class is distinguished from the Adaptability class, being narrower in scope and focusing on game content (task and scenario) modification. Its criteria are:

- **Modifying Tasks**: the ability to modify the content of an existing task, a level, its difficulty level, etc.
- **Adding Tasks**: the ability to add tasks to the game’s original ones.
- **Planning Tasks**: the ability to reorder existing or self-created tasks (where possible), to allow for the creation of a new scenario.
- **Creating Scenarios**: the ability to create a sequence of tasks.
- **Editor Provided**: whether an authoring tool is provided to support creation, modification and planning tasks.

For the purposes of our research, we introduced another class of criteria that we consider crucial: Training Ability. The purpose of this class is to identify, for instance, whether the system trains the teacher, or helps him or her to teach programming. Does the system clearly tell teachers what they can do with it? If it does, then how? Does the system provide feedback to the teachers during instructional design? Does the system provide suitable activities related to the concepts targeted? In other words, we need to pinpoint any information or means enabling teachers to get to grips with the system and to teach with it. It should be done by them either before implementing and editing a lesson in the system, or even during its previous design. In this class, we identified 4 criteria:

- **Guidelines (for playing)**: describes whether information is available that may contribute to a better understanding and a proper handling of the game.
- **Pedagogical Guidelines (for editing)**: describes whether information is available that may contribute to understanding how to teach effectively with the system.
- **Didactic Support**: describes whether there are any means to help teachers to better teach the targeted concepts (e.g. by providing suitable assignments).
- **Pedagogical Support**: describes whether there is a support system that helps teachers implement efficient courses in the game.

Thanks to the Monitoring class, we provide several criteria to describe the means made available to collect, consult and analyse logs of the learners’ actions in the game. Indeed, an adaptation of a game by teachers or trainers is often triggered by discrepancies between the expected behaviours (of the learner-players) during the design stage of the game and the actual practices that may arise during the use stage. Such discrepancies are only perceptible, in some cases, through the implementation of a monitoring system, which logs interactions between users and the game. The logs collected can be transformed in order to reveal indicators, with a high level of abstraction, that might support teachers in making the necessary adaptations to ensure the learning process runs smoothly.

This class is related to Training Ability and Adaptability (more specifically the criterion Teacher Profile). Indeed, access to the logs requires a dedicated teacher interface. The criteria of this class are:

- **Progress**: identifies the availability of logs (indicators) of learners’ progress in the game (levels completed, unlocked, levels replayed, etc.).
3 RESULTS AND DISCUSSION

Due to lack of space, we only included an excerpt of the synthetic review table of the 10 selected games in this paper in the appendix below (Table 3).

Concerning the class Adaptability, we noticed that many games have an Open Source Code (6/10) which is a positive element to support their modification. However, in the current selection, only a few games (3/10) provide specific interfaces for teachers (Teacher Profile). Among the games, the available features are very different: AlgoBlocs allows teachers to publish challenges or to enable/disable comments and forum options; AlgoPython allows teachers to create classes and assign students to them; finally, the game Code allows teachers to plan their courses, to structure them into units and chapters. The last two criteria of the class Adaptability are scarce. None of the games studied provide the possibility of modifying the game interface and only KoduGame separately, note-taking the relevant information. In a second step, a synthetic table was filled in jointly. The time allowed for the review was different for each game. Thus, we noticed that, depending on the type of game, the average time for a full review was one hour. For some games, given their simplicity or minimalist interface, the review process was significantly shorter.

The review process required more than just playing the game, it also required time for the reviewers to get to grips with it and master it: reading guides, watching tutorials before playing, and testing the level editor when available. This analysis was characterized by a “meta-playing” approach, in which the reviewers had to be aware of the game and of themselves as players during the act of playing. This attitude is necessary for the reviewers to identify pros and cons of each game and to see, as they proceed, the similarities and differences between the games.

The review was carried out by class of criteria, and as soon as all the cells in a class were filled in, the reviewer moved on to the next class in the table. For some classes, such as Adaptability and Community, the reviewer had to spend more time than others, as additional research, especially on the web. For instance, the availability of open source code, the presence of forums dedicated to the game or any additional information present on users’ blogs.

For each class, a comment column has been added to the table to provide additional information on the availability of certain criteria, but also to add specifics found in the game.

2.4 Review Process

To review each game, we proceeded as follows: first, the same game was reviewed by two reviewers separately, note-taking the relevant information. In a second step, a synthetic table was filled in jointly. The time allowed for the review was different for each game. Thus, we noticed that, depending on the type of game, the average time for a full review was one hour. For some games, given their simplicity or minimalist interface, the review process was significantly shorter.

The review process required more than just playing the game, it also required time for the reviewers to get to grips with it and master it: reading guides, watching tutorials before playing, and testing the level editor when available. This analysis was characterized by a “meta-playing” approach, in which the reviewers had to be aware of the game and of themselves as players during the act of playing. This attitude is necessary for the reviewers to identify pros and cons of each game and to see, as they proceed, the similarities and differences between the games.

The review was carried out by class of criteria, and as soon as all the cells in a class were filled in, the reviewer moved on to the next class in the table. For some classes, such as Adaptability and Community, the reviewer had to spend more time than others, as additional research, especially on the web. For instance, the availability of open source code, the presence of forums dedicated to the game or any additional information present on users’ blogs.

For each class, a comment column has been added to the table to provide additional information on the availability of certain criteria, but also to add specifics found in the game.
allows the use of interaction peripherals other than the keyboard/mouse pair.

Regarding the class editing, most games do not give any control over the scenario, some only allow the player to unlock the levels as they progress. Two games stand out: SPY and KoduGame. The first provides an option to add/remove/modify levels in the scenario by editing XML files. The latter allows teachers to create worlds in which they can define tasks, organize them, and specify assignments. Nevertheless, we noticed that in Code, tasks (called units) can be assigned to a course, but their content cannot be modified or reorganized.

For the class Training Ability, 6 games out of 10 provide guidelines and only 3 (Code, PyRates and Ceebot) are accompanied by pedagogical guidelines that provide information on the concepts covered by the game. The last two criteria in this class are dependent on the possibility to create/modify a task. Neither of the two games identified in the editing class offers means to assist the teacher on either the pedagogical or didactic aspects.

Three games stand out in the class Monitoring (AlgoPython, CodinGame and Algoblocks). The players’ actions are logged to provide teachers with dashboards displaying progress and performance indicators. Unfortunately, none of the games allows access to the data logged, preventing customized processing/analysis.

For the class CS Specific, only one game offers the player the possibility to manipulate several programming languages: CodinGame. It offers only text-based languages (no block-based languages), but several programming paradigms are possible, such as object-oriented, functional and imperative approaches. Therefore, teachers are free to choose their language from a wide range of 27 different programming languages.

Finally, the class Scenario shows us that most of the games (7/10) offer some kind of scenarios that links the different levels of the game. These links can be independent of the content of the tasks, as in PyRates, where the levels are structured by a back story including a theme and characters, but where each level can also be played independently of the other levels.

4 CONCLUSIONS

The research presented in this paper is part of the general problem of the appropriation of serious learning games dedicated to computational thinking and programming by teachers and trainers. More
ACKNOWLEDGEMENTS
The authors would like to acknowledge The University Lumière Lyon 2 for the APPI 2020 Grant.

REFERENCES
Fraser, N. (2015). Ten things we’ve learned from Blockly. 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond), 49–50.
APPENDIX

The Table 3 below shows an excerpt of the full synthetic review table. The comments and some game metadata (URL and date of visit) are omitted. The full version of the table is available online: https://recherche.univ-lyon2.fr/meta-dect/SG_adaptability_Review.xlsx

On the table, to save space we replaced “Available” with “×” and “Not available” or “No scenario” with blank cells.

Table 3: Excerpt of the full synthetic review table.

<table>
<thead>
<tr>
<th>Game</th>
<th>Spy</th>
<th>Code</th>
<th>Algopyhton</th>
<th>Pyrates</th>
<th>Collin’ Game</th>
<th>Kodu Game</th>
<th>Robocode</th>
<th>Cebot</th>
<th>Algoblocks</th>
<th>Compute it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Source Code</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teacher Profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI Modification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modifying Tasks</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adding Tasks</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planning Tasks</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating Scenarios</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editor Provided</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Ability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guidelines (for playing)</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pedagogical Guideline (for editing)</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Didactic Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pedagogical Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Progress</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background Information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log Formats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS Specific</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Forum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Author/Publisher Contact</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent</td>
<td>Small</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stand-Alone Tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>