Circle of Life: Microworld Project at the End of CS1

Sebastian Mateos Nicolajsen
CCER: Center for
Computing Education Research
IT University of Copenhagen
Denmark

Michael E. Caspersen
It-vest
&&
Aarhus University
Denmark

Claus Brabrand
CCER: Center for
Computing Education Research
IT University of Copenhagen
Denmark

Figure 1: The four (weekly) *themes* of the four-week Microworld project at the end of CS1.

Abstract

Microworlds have been proposed as an educational approach for engaging students and increasing motivation. However, the use of microworlds has largely been studied in the initial phases of learning to program (or entire courses). We adopted a microworld project at the *end* of our CS1 course to investigate its use in teaching students composition and creation of larger programs rather than initial familiarity with programming constructs. We here compare the impact of using a scaffolded, microworld project over a (previously used) non-scaffolded, non-microworld project. Based on data from N=303 students exposed to the two kinds of projects (145 in 2022 vs 158 in 2023), we compare the two cohorts according to a *test of learning (TOL)* based on the final exam grades. In 2023, we studied the impact further by considering the *feel of learning (FOL)* based on a qualitative student survey. The results show that the student grade-point average (TOL) increased significantly when we adopted the microworld project. Further, the microworld project appears to produce a high FOL.

CCS Concepts

• **Social and professional topics** → **Computing education**; **Computing education**; • **Applied computing** → **Education**.

Keywords

Introductory programming, project, microworld, scaffolding.

ACM Reference Format:
Sebastian Mateos Nicolajsen, Michael E. Caspersen, and Claus Brabrand. 2025. Circle of Life: Microworld Project at the End of CS1. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701808

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701808

1 Introduction

Several studies suggest that learning to program is a difficult endeavour due to a lack of motivation, complex programming elements, and missing scaffolding which may in turn lead to more stress and worse mental health of students [23, 36, 37, 52]. The computing education community, therefore, attempts to make programming more interesting, relevant, and effective [26]. One approach to this is using *microworlds*, known for engaging and facilitating complex concept understanding [15].

However, the use of microworlds have largely been studied in the initial phases of learning to program (or entire courses). Here, we investigate the application of microworlds at the *end* of a CS1 course to investigate how a microworld may be used to improve students' ability to compose and create larger programs rather than learning programming constructs initially. Specifically, we present a four-week project used at the *end* of our introductory programming (CS1) course wherein students in groups of three: *design*, *implement*, and *test* an ecosystem of animals, plants, and fungi using our own 'microworld' library (in Java).

The project scaffolds initial experimentation prior to project start and progresses with weekly open-ended requirement themes. Specifically, we designed the project around *constructionism*:

> *"**Constructionism**—the N word*[1] *as opposed to the V word—shares constructivism's connotation of learning as 'building knowledge structures' irrespective of the circumstances of the learning. It then adds the idea that this happens especially felicitously in a context where the learner is engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe."*
>
> — Papert & Harel, 1991 [43]

Specifically, we explore the impact of a microworld project at the *end* of a CS1 course within our Bachelor's programme, *Software*

[1] By this, Papert & Harel meant "ConstructioNism" rather than "constructiVism."

development at the IT University of Copenhagen. We compare the student cohort of 158 students exposed to the scaffolded microworld project (in 2023) vs the previous cohort of 145 students who were exposed to a non-scaffolded non-microworld project (in 2022) using a *test* of their learning (TOL) [12]. Further, we support these findings by investigating the *feel* of learning (FOL) from the cohort exposed to the microworld project.

These cohorts were exposed to the same course material, lectures, exercises, mandatory activities, and exam. The only difference was the *project* they were exposed to. The previous cohort was given an open-ended project with a set of functional requirements at the start of the project and nothing more, whereas the 2023 cohort was given a scaffolded microworld project.

2 Background

Several approaches aim to address the challenges associated with learning to program. Among these are educational programming environments, games, and microworlds [8, 25, 37]. These approaches have evolved rapidly over the past years due to students being more familiarized with computer games [27]. We outline literature exploring microworlds, educational games, educational environments, and various attributes of creating collaborative learning.

2.1 Microworlds

The term *'microworld'* is argued to originate from the SHRDLU program [15, 56][2] but it was popularized by Papert's use in relation to the LOGO programming language wherein a set of commands allow the user to control a turtle [44, 49]:

> *"a subset of reality or a constructed reality whose structure matches that of a given cognitive mechanism so as to provide an environment where the latter can operate effectively. The concept leads to the project of inventing microworlds so structured as to allow a human learner to exercise particular powerful ideas or intellectual skills."*
>
> — Papert, 1981 [42]

Here, Papert coined the term *constructionism*, which has evolved beyond its roots in constructivism by specifically emphasising that the learner is engaged in constructing some public entity [43, 49].

Rieber [49] argues that microworlds are difficult to distinguish and are often associated with other terms; e.g., computational media [13], interactive simulations [53], and computer-based manipulatives [31]. Computer-based microworlds, however, enable more people to understand complex systems and their principles by allowing qualitative understanding through the building and use of concrete models [49]. To some extent, microworlds are "embodiments" of ideas, and, therefore, their benefits lie in the way they represent a problem for students, making students capable of understanding the problem [18, 45]. Thus, microworlds establish a strong semantic link between something formal and abstract and something real or simulated [7]. Microworlds, therefore, allow the student to perform the necessary explanation, justification, and prediction related to the problem(s) under scrutiny [45]. Microworlds are often assumed to be simulations due to the concepts being difficult to distinguish [15, 17, 49]. However, the main difference is that a microworld serves novices an entryway into understanding the underlying model [15, 17]. Thus, a simulation is not necessarily a microworld [15]. Further, a microworld is not necessarily a simulation of anything but instead represents a model as is [49].

In the context of programming languages, microworlds decrease the distance between student mental models and the programming language [58], typically via the use of a graphical scene [6, 10, 30].

[2]First used in an internal MIT report co-authored by Papert in 1972 [39].

2.2 Visualization

Microworlds do not necessarily possess a graphical or visual interface, but visual tools have aided students to grasp various scientific and programming-specific concepts through familiar representations [10, 33, 34, 52]. Visualizations may be internal to the individual (mental images), or external [15]. Both these types of visualizations depend on the use of metaphors and analogies due to individuals often associating new concepts in terms of something familiar [22]. Such visual metaphors are argued to be more effective than verbal ones, including those given through a lecture [3]. Thus, programming microworlds are typically created with visual metaphors to increase accessibility for novices, as novices have yet to develop the skills to visualize program code themselves [15]. Such microworlds, therefore, visualize command execution through changes to the visible state [10, 15, 34, 58]. Specifically, research has found that one of the most important aspects of microworlds in object-oriented programming is the visualization of classes and objects [59].

2.3 Educational Game Theory

In combination with visualizations, microworlds typically employ game elements, design, and techniques as a way to structure the learning of novices [1, 48]. Educational games have also been used to create an attractive and interactive learning context to motivate students, with positive impact [21, 33, 37, 38]. Educational games are popular across gender, ethnicity, and socioeconomic status [38]. Such games positively develop performance in the class room [9, 60], creativity [40], and motivation [11].

There are multiple frameworks for designing educational games. Malliarakis et al. [37] provide an overview of these: *four-dimensional framework*, *conceptual framework for educational games*, and *design, play, and experience framework*. Largely, they reinforce the values associated with the broader approach of *exploratory learning* to which microworlds belong [49]. Specifically, educational games employ *fantasy* to encourage learners to imagine activities in an imaginary world [15, 19], *challenges and curiosity* to educate and foster intrinsic motivation [15, 19, 48], and they support ways of providing instantaneous visual feedback [15].

2.4 Educational Environments

Microworlds can be distinguished from educational (programming) environments in their typical focus on immersive learning of specific parameters and components of programming [58]. However, when applying the structural view of microworlds, Xinogalos et al. argue that any software sharing the physical and design attributes can rightfully be called a microworld [59]. Despite their positive impact on novice learning, we argue that development environments are *not* microworlds. However, acknowledging the significant importance of programming environments. For example, we use BLUEJ in our CS1 course [35].

2.5 Project work

Project-based learning (PBL) affords authentic learning tasks [24]. While there are different definitions of PBL, Grant argues the critical components to be: 1) a driving question or problem; *and* 2) the production of artefact(s) as representations of learning [24]. PBL is inherently connected to constructionism as the goal is the production of an artefact. This is, in fact, also what distinguishes it from problem-based learning, despite many similarities [24, 29]. Interestingly, the knowledge acquired during the production of the artefact can be regarded as more important than the produced artefact itself [24]. In addition, PBL fosters self-direction and self-regulation and thereby offers the potential to integrate meaningful collaboration and cooperation [24].

2.6 Scaffolding

Scaffolding originates from its use in building constructions to allow workers to access and rely on the structure as it is constructed. Scaffolding was used in education as a metaphor to capture the nature of *support and guidance* in learning [28]. It has further been used to describe the act of building upon what students already know to support them in learning something they do not [51]. Scaffolding can also be seen as a process wherein the educator controls the elements of a task which is initially beyond the learner's capacity [57]. The goal of the guidance that scaffolding enables is to fade the support gradually until the learner becomes independent [51]. While there are various scaffolding approaches, in this study, we apply a 'fixed' and non-negotiable scaffolding as we provide a microworld, which students are required to apply in the project [50]. Research also suggests that learners perceive scaffolding differently, depending on its representation [51]. Further, scaffolding works well in the context of self-regulated learning [51].

3 Related Work

There are many microworlds for programming learning, but we have not found literature examining the effect of microworlds in a *final* project (when students know *some* programming).

We already discussed Logo, but there are several later microworlds inspired by (the limitations) of Logo; e.g., StarLogo and Netlogo supporting multiple actors, used to model agent-based behaviour [44, 47, 54, 55]. Some supporting multiple characters (sprites), different sounds and images to foster creativity [55]. Similarly, Boxer, a multipurpose computational medium, was created for the same reasons, integrating various computational representations [14].

Other examples include Scratch, wherein students can create interactive stories and games through a drag-and-drop interface [46]. Originally intended to support students aged 10 to 18 through the availability of different characters, backgrounds, sounds, and images. However, microworlds do not necessarily focus on games [41]. Several specific microworlds also support computational problem-solving in different domains, e.g., Genscope [32].

In addition to the 2-dimensional microworlds mentioned here, there exists a plethora of microworlds combining elements of game engines in 3-dimensions with the purpose of teaching (different) programming languages; e.g., Alice [10], CMX [38], ProGo [15], Karel [5, 58], and Meadow [2]. Some of these combine multiplayer aspects [38] while others focus on realistic graphics representation of object-oriented programming concepts [15]. Finally, there exist examples of embedding programming learning *into* games, e.g., the "hour of code" in Minecraft [20].

4 The 'Microworld' Project

Our microworld project is inspired by a 2-dimensional programming exercise entitled "*Foxes & Rabbits*" [4]. During the design of our microworld (illustrated in Figure 2), we incorporated the theory regarding microworlds, visualization, and educational game theory. In particular, the microworld visualizes a small floating rock where different actors can exist (visualized through an isomorphic 2.5-dimensional view). The microworld simulates a day-night cycle to enable actors to change behaviour according to the time of day. The microworld framework is implemented in Java, as this is the programming language employed in the course.

Students are expected to implement *actors*, which they can insert into a *world*. Students then utilise the abstractions of the *world* to implement certain behaviours; e.g., move to another location, retrieve adjacent actors, and delete certain actors. By implementing a specific interface, each actor is simulated within our world (by recurringly invoking methods implemented in student classes). Figure 2 illustrates one example involving a rabbit and a wolf. Originally, there were two rabbits; the wolf, however, killed one rabbit, and a carcass is now left behind.

While this paper is limited in terms of what we show in regards to the microworld library, the code (and associated documentation), an introductory guide to get students started, and the full set of weekly requirements are accessible here.[3][4] We expect that others can easily adapt the project given the accessible documents, or start anew using the contents of the repository. Given that the majority of our work consisted in the creation of these resources, adaption and execution should yield only minor to no additional overhead from an instructor perspective. Comparing the microworld project with the previous project, we see no increase in instructor time, in fact, we see a decrease as much of the assessment has been *integrated* into existing teaching assistant tasks.[5]

The microworld provided thus depicts a small grid-like rocky world wherein actors can exist and behave. The microworld simulates actions by creating small animations to indicate 1) *appearance*,[6] 2) *movement*, and 3) *disappearance*. Students can then bind creative homemade images (aka, *sprites*) to the actors or use predefined ones (and even change these during runtime to animate certain behaviour). Thus, once students implement actors, they come to life through their respective sprites, and their implemented logic becomes visual through animated behaviour.

As the microworld is employed by students who have already been introduced to Java, it is presented as a new library where the students have to explore the documentation to figure out what is feasible and what is not. Thus, the purpose of the microworld is not to scaffold initial learning of programming but rather scaffold development of larger programs and enable students to "visualise" their

[3] https://github.com/sebastiannicolajsen/WorldSimulator
[4] Note that the associated documents (with the exception of the code) are machine translated from danish.
[5] As the TAs were already doing similar tasks, we have seen little to no overhead from their perspective.
[6] When an actor first manifests itself or resurfaces (e.g., a rabbit exiting its burrow).

Figure 2: An image of a rabbit, a wolf, and the carcass left over from a dead rabbit during daytime (10 iterations into the simulation).

code as behaviour through simulation when developing a larger program. Further, it is designed to encourage students to create and manipulate objects (actors) to discover otherwise unexpected (more complex) behaviour. By animating the simulation, we hope to foster creativity through familiar metaphors [16, 18, 34].

The microworld project attempts to combine the theory of project work, scaffolding, and self-regulated learning through the use of the mentioned microworld. Thus, the project requires students to *design*, *implement*, and *test* various different animals, plants, and fungi according to a set of requirements.

4.1 Microworld vs Non-Microworld Project.

In 2022, our CS1 students did a non-scaffolded non-microworld project. While the new 2023 scaffolded microworld project is largely similar in format (in terms of duration, group size, requirement scope, and group-work training), there were also differences. Table 1 highlights similarities and differences. Further, the course was taught by the same instructors. We here elaborate on the primary differences.

Weekly Theme. The preivous project had no weekly themes. However, with the microworld project students, each week, receive a new *theme* comprising a set of loosely connected requirements that they must adhere to. The themes are connected through an overarching theme being the *"Circle of Life."* Figure 1 provides an overview of the four (weekly) themes: Theme 1 (week 1), is about *plants and herbivores* (specifically, students must implement grass and rabbits); Theme 2 is on *carnivores and predation* (wolves and bears); Theme 3 relates to *fungi and decomposition* (mushrooms); and the last week (Theme 4) is a *student-chosen theme* (potentially including fictional animals). For realism, we had a *domain expert* (biologist) assess the requirements of each theme and provide students with insights on how these animals, plants, and fungi behave in the real world (if requirements deviated too far from reality); exemplified by the following objection to a *bear (capable of) "hunting" wolves*:

> *"A biologist would presumably not describe this as 'hunting,' since a bear's diet is 73% plants, 22% insects, and 5% animals (according to a study of black bears)."*
>
> — Quote from domain expert (Biologist).

Requirement Specification. In extension of the above, a significant difference between the projects is how students received requirements. For the previous project, students received a complete list of the (functional) requirements that their system should adhere to at the beginning of the project. For the microworld project, students received these as part of each theme — simulating an 'agile' approach to development. Thus, the original project encourages and typically follows a waterfall approach to development, for which the students performed *analysis* (of requirements), *design*, *implementation*, and *testing*. Instead, the microworld project scaffolds this process and forces an iterative and incremental approach to development as the students do not know all the requirements at the beginning [51]. Albeit the latter requirements are related to adding new actors, they encourage and enforce iteration upon earlier abstractions (decisions made by the students).

Project Goals. The previous project required students to *analyze*, *design*, *implement*, and *test* a streaming service wherein you can load movies from a text file, show these in a graphical user interface, sort them according to various attributes, and save favourites. However, the service should not be capable of actually streaming movies. The microworld project instead focuses on students *designing*, *developing*, and *testing* various different animals, plants, and fungi to behave according to given individual requirements.

Starter guide & base project. The previous project had little to no scaffolding. Specifically, students were provided a general introduction to the project and suggestions for designing the system at large. However, students were required to start from nothing. The microworld project also provided a general introduction to the project but further supplied students with a base project (containing the library) and a starter guide wherein students develop a simple actor and tests thereof. Further, the guide elaborates on some initial issues students may encounter.

Reports. The previous project required students to deliver a final report containing their analysis of requirements, design decisions, and reflections. The microworld project requires the same (albeit in a shorter format). Further, the microworld project required weekly reports on the same topics as input for supervision to encourage continuous reflection on decisions. The idea behind this change is also to give students more opportunities to get feedback on (smaller parts of) their writing.

Graphical User Interface Programming. A significant part of the previous project (up to half of the delivered code) was GUI-related. In the microworld project, students delivered no graphical components as the library produced these. Despite this difference, delivered code bases were of similar size. In other words, the microworld project involves (trains) a lot more non-GUI programming.

5 Context

The project is situated at the *end* of an introductory programming (CS1) course, which is a mandatory 15 ECTS[7] first-semester course on the Bachelor of Software Development (CS majors) at the IT University of Copenhagen. The course is divided into two phases. The first phase of the course lasts for 11 weeks and focuses on teaching imperative and object-oriented programming in Java and BlueJ [35]. The students have three mandatory one-hour on-site

[7] One academic year is 60 ECTS (European Credit Transfer and Accumulation System.

Property	Previous project	Microworld project
Topic	Streaming service	Life simulation
Development model	Waterfall	Iterative
Year	2022	2023
Duration	4 weeks	4 weeks
Group size	3 students	3 students
Group formation	Student formed	Student formed
Project & group-work training	Yes (Separate course)	Yes (Separate course)
Unit tests required	Yes	Yes
Weekly additions	No	Yes
Requires GUI programming	Yes	No
Starter guide & base project	No	Yes
Requirement specification	Once (beginning)	Weekly
Supervision with TAs	By need	Weekly
Reports	Final report	Weekly & Final reports

Table 1: Comparison: previous project vs microworld project.

Number of students	**Previous project**	**Microworld project**
Groups	45	51
Enrolled in CS1	160	172
Attended exam	**145**	**158**
Passed exam	126	150
Failed exam	19	8

Table 2: Number of *students* and *groups* in 2022 vs 2023.

programming tests and two assignments that have to be passed to be eligible for the exam. The second phase of the course lasts four weeks; here, the students undertake the project—the topic of this paper—in groups of three. The project intends to *train* student programming competences in a prolonged, practical context. The project will thus impact the programming competences of the students and, thereby, the final course exam grades.

Students. The course enrols around 150–175 students each year; whereof around 20% are women and the median age is 21 (years). Table 2 compares the number of students and groups in 2022 vs 2023 (i.e., *without* vs *with* a scaffolded microworld project). The number of enrolled students is larger than three times the number of groups because of students *re*-taking the exam or course. (An already completed-and-submitted project is valid for subsequent examination attempts in that a student does not have to *re*-do the project if they did not attend or failed an earlier exam.)

Exam. The course ends with a 30-minute individual oral exam which comprises several parts: First, students receive a *random topic* (e.g., *inheritance* or *exceptions*) from a set of eight topics announced a month before the exam. Students then have to explain the topic on a whiteboard while the examiners are asking questions. Second, students perform *live programming* tasks on their own computer while explaining what they are doing. This is based on a *simplified version* of the project (four classes, each of about 10–30 lines) that the students are asked to modify in various ways (e.g., change an `ArrayList` to a built-in `array`, reverse the traversal order of a `for` loop, and/or add appropriate exception handling). Third, the project has been submitted and assessed by the examiners. Students receive *one grade*, which encompasses the "overall performance" on all parts of the exam, i.e., *explanation*, *live programming*, and *project*.

6 Methodology

To evaluate the effect of our scaffolded microworld project in comparison to the previous non-scaffolded project, without any microworld (application development), we compare students according to a *Test of Learning (TOL)* based on individual student course grades. Further, we explore students' affective responses regarding the Scaffolded microworld project (their *Feel of Learning (FOL)*) based on a survey administered to the students [12].

6.1 Objective

Specifically, we investigate two research questions:

> **RQ1 (TOL):** To what extent did the students learn from undertaking the microworld project, according to a *test of learning* (final exam course grades)?
>
> **RQ2 (FOL):** To what extent did the students *feel* they learned from undertaking the microworld project (based on survey responses)?

6.2 Metrics

We elaborate the two metrics involved, namely *TOL* and *FOL* [12]:

Test of Learning (TOL). In Denmark, exams are conducted by (at least) two educators; the course teacher acts as the *internal* examiner and is accompanied by an *external* examiner (aka, *censor*) from another independent Danish university. The internal and external examiners must agree on the grade; in case of disagreement, an average of the two verdicts is computed and rounded off in the direction of the external examiner. This independent oversight improves the objectivity and reliability of comparing the two student cohorts to determine how the project affects student TOL (as measured by the final exam grades).

Feel of Learning (FOL). We conducted a survey to identify the affective opinions of the students on the Microworld project. Specifically, we asked students using a Likert scale from 1–5 (where 1 indicates strong disagreement and 5 indicates strong agreement) whether they *felt*: 1) like they got better at programming; 2) properly challenged; 3) prepared for the next semester; and 4) the project was fun. Students were encouraged to expand on anything related to the project (both positive and negative).

6.3 Analysis

For comparing *vectorized* data (grades), we will use the Mann-Whitney U-test (aka, Wilcoxon Rank-Sum). For comparing *ratios*, we use a Z-test (for two population proportions), i.e., whether x/x' is a statistically significantly larger proportion than y/y'. For all statistical analyses, we adopt a 95% confidence interval (with α=5%) to designate *statistical significance* and use (symmetric) two-tailed tests, not presupposing the superiority of either testee. For the survey results, we present students' quantitative responses and outline tendencies from students' qualitative responses.

7 Results

We consider the results according to our two research questions:

Test of Learning (TOL). Figure 3 shows a comparison of the distribution of the grades under the old unscaffolded project (in 2022) vs the new scaffolded microworld project (in 2023). In 2022, 145 students attended the final exam compared to 158 students in 2023. In 2022, the grade-point average among the students was **7.03**, which increased to **7.63** in 2023. (In Denmark, grade averages are

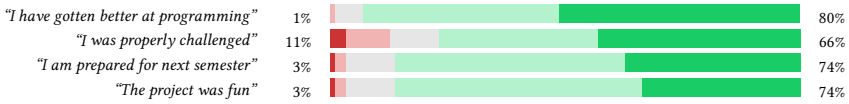

Table 3: Overview of Likert responses: strongly disagree; disagree; indifferent; agree; strongly agree.

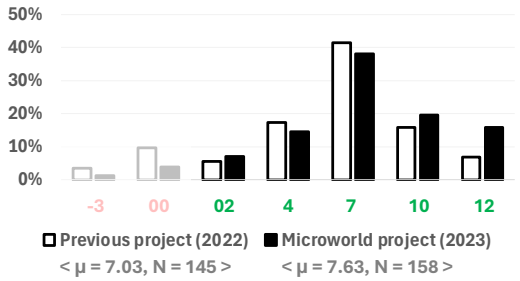

Figure 3: Comparison of the grade distributions under the previous project (2022) vs the microworld project (2023). Fail grades (-3 & 00) are shown in red; pass grades (02–12) in green. Averages are of passed grades only (customary in Denmark).

computed from *pass* grades only.) Comparative analysis of all grades (pass and fail) reveals that the difference (+8½% grade increase) is *statistically significant* (p=0.0076), according to a U-test. In 2022, **13%** (19 out of 145) *failed* the final exam; in 2023, this ratio dropped to **5.1%** (8 out of 158). This difference (failure decrease) is *statistically significant* (p=0.014), according to a Z-test.

Following the 2023 exam, the internal and external examiners involved in both years discussed and noted that, in 2023, the students were generally better specifically at the *live programming* part of the exam (in comparison to the year earlier). We speculate that this improvement is due to *improved training of relevant programming competences* during the scaffolded microworld project.

Feel of Learning (FOL). When comparing the two projects through the official course evaluation (on a scale from 1 to 6, where higher is better), there were no significant differences in student evaluation of the two instances: In 2022, the average of the course evaluations was **5.62** (from 53 students); in 2023, it was similar at **5.66** (from 70 students). This difference is not significant (p=0.697), according to a U-test.

In 2023, we conducted a more in-depth survey focusing on students' perception of the scaffolded microworld project. For this, we obtained responses from 55% of students (87 students of 156 students). Table 3 summarises the students' quantitative responses regarding the project.

From this, it is apparent that the project had a positive influence on students' *feel* of learning. In particular, the majority of students felt they 1) got better at programming, 2) were properly challenged, 3) prepared for the next semester, and 4) found the project fun. These findings largely reflect what we found examining students' grades. Within the responses, 34 students particularly highlighted the value of the themes and the project as a whole, e.g., the value that the visualization/microworld provides [6, 10]:

> *Projects like this are fun, and you can relate it to reality on some level — It makes the classes we implement make a lot of sense.*

Further, the students also highlight the value of continuous handout of requirements [51]:

> *I think it worked really well with the theme system so that you were constantly building up your simulation more and more each week instead of maybe just giving all the requirements at once. It feels that there was a very good progression in the project, and it made sense what was added each week.*

However, 27 students also highlighted that the requirements could have been distributed more evenly between each week, as they felt some weeks were overloaded compared to some of the others. Last, 15 students highlighted the benefit of supervision each week.

8 Threats to validity

The main threat to validity involves a **change of teaching format.** In 2022, we changed the teaching format from the classic *lectures-followed-by-exercises* (2 hours of lecture + 2 hours of exercise classes, twice per week) to *zipped* lectures & exercises (4 hours of interleaved lectures and exercises, twice per week) [anonymized]. However, since this change was already in effect in 2022 under the last edition of the old project, we expect minimal impact from this change.

9 Conclusion

Microworlds, scaffolding, and problem-based learning has been applied in introductory programming education in attempts to improve interest, relevance, and effectiveness of teaching [15, 26, 52]. Largely, microworlds have been studied in the initial phases of learning to program (or entire courses). In this study, we developed and integrated a scaffolded microworld project at the *end* of an introductory programming (CS1) course. Specifically, we designed a four-week project wherein students work in groups of three *design*, *implement*, and *test* an ecosystem of animals, plants, and fungi using our own 'microworld' library (in Java).

We compared this project to a previous, non-scaffolded, project which did not utilize a microworld. Specifically, we investigated the effect on students learning (through a test of learning) and their affective responses to the project (feel of learning). We found that the microworld project significantly increased students grade-point average from **7.03** (in 2022) to **7.63** (in 2023). Further, students responded positively to the project, enjoying the structured distribution of requirements and the project's overarching theme.

In conclusion, we argue that microworlds are thus not *exclusively* introductory tools. They can also enhance the learning process once students have been taught *some* programming so as to train and hone relevant programming competences during a project.

10 Acknowledgement

A special thanks to Mathias Fink and Anders Wagner for help with implementation and Lucas Geisler for providing us with the domain expertise.

References

[1] Marco Aedo Lopez, Elizabeth Vidal Duarte, Eveling Castro Gutierrez, and Alfredo Paz Valderrama. 2016. Teaching Abstraction, Function and Reuse in the first

class of CS1: A Lightbot Experience. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 256–257.
[2] Eike Falk Anderson and Leigh McLoughlin. 2007. Critters in the classroom: a 3d computer-game-like tool for teaching programming to computer animation students. In *ACM SIGGRAPH 2007 educators program*. 7–es.
[3] Lynne P Baldwin and Jasna Kuljis. 2001. Learning programming using program visualization techniques. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. IEEE, 8–pp.
[4] David John Barnes, Michael Kölling, and James Gosling. 2006. *Objects First with Java: A practical introduction using BlueJ*. Pearson/Prentice Hall.
[5] Byron Weber Becker. 2001. Teaching CS1 with karel the robot in Java. In *Proc. 32nd SIGCSE technical symposium on Computer Science Education*. 50–54.
[6] Joseph Bergin, Mark Stehlik, Jim Roberts, and Rich Pattis. 2005. *Karel J. Robot: A gentle introduction to the art of object-oriented programming in Java*. Dream Songs Redwood City.
[7] Eric Bruillard. 1997. *Les machines à enseigner*. Hermès Paris.
[8] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, Anatoly Kouchnirenko, and Philip Miller. 1997. Mini-languages: a way to learn programming principles. *Education and information technologies* 2 (1997), 65–83.
[9] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. 2009. Experimental evaluation of teaching recursion in a video game. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*. 79–86.
[10] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of computing sciences in colleges* 15, 5 (2000), 107–116.
[11] Sara De Freitas and Steve Jarvis. 2006. A framework for developing serious games to meet learner needs. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*.
[12] Louis Deslauriers, Logan S. McCarty, Kelly Miller, Kristina Callaghan, and Greg Kestin. 2019. Measuring actual learning versus feeling of learning in response to being actively engaged in the classroom. *Proceedings of the National Academy of Sciences* 116, 39 (2019), 19251–19257.
[13] Andrea diSessa. 1989. *Computational media as a foundation for new learning cultures*. Technical Report. Boxer Group Technical Report-G5, Berkeley, CA.
[14] Andrea A DiSessa. 1991. An overview of Boxer. *Journal of Mathematical Behavior* 10, 1 (1991), 3–15.
[15] Fahima Djelil, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi, Eric Sanchez, and Jean-Marc Lavest. 2016. Microworlds for learning Object-Oriented Programming: Considerations from research to practice. *Journal of Interactive Learning Research* 27, 3 (2016), 247–266.
[16] Mary Jo Dondlinger. 2007. Educational video game design: A review of the literature. *Journal of applied educational technology* 4, 1 (2007), 21–31.
[17] Laurie D Edwards. 1995. Microworlds as representations. In *Computers and exploratory learning*. Springer, 127–154.
[18] Laurie D Edwards et al. 1995. Computers and exploratory learning. (1995).
[19] Rosemary Garris, Robert Ahlers, and James E Driskell. 2002. Games, motivation, and learning: A research and practice model. *Simulation & gaming* 33, 4 (2002), 441–467.
[20] Reza GhasemAghaei, Ali Arya, and Robert Biddle. 2017. Affective walkthroughs and heuristics: Evaluating minecraft hour of code. In *Learning and Collaboration Technologies. Technology in Education: 4th International Conference, LCT 2017, Proceedings, Part II 4*. Springer, 22–40.
[21] Andreas Giannakoulas and Stelios Xinogalos. 2018. A pilot study on the effectiveness and acceptance of an educational game for teaching programming concepts to primary school students. *Education and Information Tech* 23 (2018), 2029–2052.
[22] John K Gilbert. 2010. The role of visual representations in the learning and teaching of science: An introduction.. In *Asia-Pacific Forum on Science Learning & Teaching*, Vol. 11.
[23] Anabela Gomes and António José Mendes. 2007. An environment to improve programming education. In *Proceedings of the 2007 international conference on Computer systems and technologies*. 1–6.
[24] Michael M Grant. 2011. Learning, beliefs, and products: Students' perspectives with project-based learning. *Interdisciplinary Journal of problem-based learning* 5, 2 (2011), 6.
[25] Glenda A Gunter, Robert F Kenny, and Erik H Vick. 2008. Taking educational games seriously: using the RETAIN model to design endogenous fantasy into standalone educational games. *Edu. Tech. Research & Dev.* 56 (2008), 511–537.
[26] Mark Guzdial. 2002. Programming Environments for Novices. *Retrived from novice-envs2. pdf (umich. edu)* (2002).
[27] Mark Guzdial and Elliot Soloway. 2002. Teaching the Nintendo generation to program. *Commun. ACM* 45, 4 (2002), 17–21.
[28] Jennifer Hammond. 2001. *Scaffolding: Teaching and learning in language and literacy education*. ERIC.
[29] I Harel and S Papert. 1991. Software design as a learning environment. Constructionism. Norwood.
[30] Poul Henriksen and Michael Kölling. 2004. Greenfoot: combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. 73–82.
[31] Paul Horwitz. 2000. Hypermodels: Embedding curriculum and assessment in computer-based manipulatives. In *International Conference on Mathematics/-Science Education and Technology*. Association for the Advancement of Computing in Education (AACE), 207–213.
[32] Paul Horwitz and M Christie. 2000. Computer-based manipulatives for teaching scientific reasoning: An example. *Innovations in science and mathematics education: Advanced designs for technologies of learning* (2000), 163–191.
[33] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan MacKinnon. 2012. Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science* 9 (2012), 522–531.
[34] Michael Kölling. 2010. The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–21.
[35] Michael Kölling and John Rosenberg. 2002. BlueJ-The Hitch-Hikers Guide to Object Orientation. (2002).
[36] Savannah Lopez, Jeremy Hsu, and Patricia Halpin. 2023. Students prefer final projects that are scaffolded to avoid procrastination and negative impact on mental health. *Physiology* 38, S1 (2023), 5731598.
[37] Christos Malliarakis, Maya Satratzemi, and Stelios Xinogalos. 2014. Designing Educational Games for Computer Programming: A Holistic Framework. *Electronic Journal of e-Learning* 12, 3 (2014), 281–298.
[38] Christos Malliarakis, Maya Satratzemi, and Stelios Xinogalos. 2016. CMX: The effects of an educational MMORPG on learning and teaching computer programming. *IEEE Transactions on Learning Technologies* 10, 2 (2016), 219–235.
[39] Marvin Minsky and Seymour A Papert. 1972. Artificial intelligence progress report. (1972).
[40] Jackie O'Kelly and J Paul Gibson. 2006. RoboCode & problem-based learning: a non-prescriptive approach to teaching programming. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. 217–221.
[41] Stamatios Papadakis, Michail Kalogiannakis, Vasileios Orfanakis, and Nicholas Zaranis. 2014. Novice programming environments. Scratch & app inventor: a first comparison. In *Proceedings of the 2014 workshop on interaction design in educational enviraonments*. 1–7.
[42] Seymour Papert. 1981. Computer-based microworlds as incubators for powerful ideas. *The computer in the school: Tutor, tool, tutee* (1981), 203–210.
[43] Seymour Papert and Idit Harel. 1991. Situating constructionism. *constructionism* 36, 2 (1991), 1–11.
[44] Seymour A Papert. 2020. *Mindstorms: Children, computers, and powerful ideas*. Basic books.
[45] David N Perkins and Chris Unger. 1994. A new look in representations for mathematics and science learning. *Instructional Science* 22, 1 (1994), 1–37.
[46] Mitchel Resnick. 2007. All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*. 1–6.
[47] Mitchel Resnick. 2012. New paradigms for computing, new paradigms for thinking. In *Constructionism in practice*. Routledge, 255–267.
[48] Lloyd P Rieber. 1996. Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational technology research and development* 44, 2 (1996), 43–58.
[49] Lloyd P Rieber. 2013. Microworlds. In *Handbook of research on educational communications and technology*. Routledge, 582–602.
[50] Priya Sharma and Michael J Hannafin. 2007. Scaffolding in technology-enhanced learning environments. *Interactive learning environments* 15, 1 (2007), 27–46.
[51] Clare Stanier. 2015. Scaffolding in a higher education context. In *ICERI2015 Proceedings*. IATED, 7781–7790.
[52] Spyros Vosinakis, George Anastassakis, and Panayiotis Koutsabasis. 2018. Teaching and learning logic programming in virtual worlds using interactive microworld representations. *British J. of Educational Technology* 49, 1 (2018), 30–44.
[53] Barbara Y White. 1992. A microworld-based approach to science education. In *New directions in educational technology*. Springer, 227–242.
[54] Uri Wilensky and William Rand. 2015. *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. Mit Press.
[55] Uriel Joseph Wilensky. 1993. *Connected mathematics: builiding concrete relationships with mathematical knowledge*. Ph. D. Dissertation. MIT.
[56] Terry Winograd. 1972. Understanding natural language. *Cognitive psychology* 3, 1 (1972), 1–191.
[57] David Wood, Jerome S Bruner, and Gail Ross. 1976. The role of tutoring in problem solving. *Journal of child psychology and psychiatry* 17, 2 (1976), 89–100.
[58] Stelios Xinogalos, Maya Satratzemi, and Vassilios Dagdilelis. 2006. An introduction to object-oriented programming with a didactic microworld: objectKarel. *Computers & Education* 47, 2 (2006), 148–171.
[59] Stelios Xinogalos, Maya Satratzemi, and Christos Malliarakis. 2017. Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment? *Education and Information Technologies* 22 (2017), 145–176.
[60] Amri Yusoff, Richard Crowder, Lester Gilbert, and Gary Wills. 2009. A conceptual framework for serious games. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*. IEEE, 21–23.