Universiteit Leiden
The Netherlands

Opleiding Informatica

An interactive program visualisation tool for Hedy

Ali Esat Özbay

Supervisor:
Dr.ir. F.F.J. Hermans

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

23/6/2022

Abstract

Hedy is a gradual programming language for young students, with an approach that makes learning syntax easy. Although this addresses some of the challenges in learning to code, others remain. Misconceptions in particular form a significant obstacle in a student's conceptual understanding. In a study with primary school students, We have investigated the extent to which a game-like interactive program visualisation tool can help resolve misconceptions. We found both promise and indications of the limits of such a tool, and implications for a fruitful avenue for Hedy to improve as a learning system.

Contents

1 Introduction

It may surprise the reader how young some children are today when they start to learn programming. In some elementary schools children are as young as 10 years old when they encounter programming in the classroom[6].

One reason for this is new attention given to our increasingly digital society. Policymakers all over the world attempt to meet new demands by integrating computer science into education curriculum[4].

A second reason is that programming education has evolved to the point that learning to program has never been so easy or accessible for young people. Initiatives like Scratch have reached a staggering number of children and inspired them to make creations of their own[3]. These initiatives make it possible to introduce programming so early in the classroom.

Hedy is such an initiative, an open-source gradual programming language that makes it easier of children to learn to code[1]. It's available for anyone to use online, and currently undergoes active development. Learning to code presents many challenges. One such challenge is needing to remember a lot of detailed rules right at the start. This is especially difficult for children. Hedy removes the need to remember detailed rules at first, so that learners can write simplified but meaningful code right from the start. It then gradually builds up the level of detail in so called levels, until children can finally write Python code in the last level.

Another challenge are misconceptions. Misconceptions are errors in conceptual knowledge. Conceptual knowledge is about knowing how programming concepts work and "what happens inside the computer"[5]. The current study presents an interactive program visualisation tool to help resolve such misconceptions, Hedy's first. We have investigated how effective the tool is in a study with eight Dutch primary school students familiar with Hedy. In doing so, we sought to answer the question: "To what extent can an interactive program visualisation tool help to resolve multiple variable assignment misconceptions in young users of Hedy for level 2 example code?"

We have found that the tool we developed can be both an effective and welcomed way to resolve misconceptions. The presented tool is a significant first step in addressing misconceptions for Hedy, and points to a promising direction towards making Hedy a more complete learning system.

2 Background and Motivation

Hedy [1] is a gradual programming language meant to teach young students how to program. It's available for anyone on the internet, and it is actively being developed by people from all over the world. New features are frequently added, and it's already been translated to many languages by international volunteers. In fact, anyone can contribute to the project because it's open-source. Besides being freely available online, Hedy is also taught in some primary schools, and it's growing more popular.

2.1 Gradual

Where other programming languages give learners access to all its features right from the start, Hedy takes a gradual approach. Hedy only introduces one or two new ideas at a time, sectioned into what are called levels. For example, in level one learners learn the print command, which they can use to make message appear on the screen. However, in level two they can also save their messages in variables. Instead of having to type the whole message into the print command, they can write the message into a variable. Then, learners only have to type the variable's name into the print command.

Learning to program is hard, so Hedy hides unnecessary, complex details early on to make it easier. These are syntax details, the rules on how code should be written, so we say that Hedy defers syntactic complexity. For example, learners do not need to put quotation marks around their messages in level one, but this becomes required starting in level four. Hedy continues to introduce new ideas that start out simple, and get more detailed later on, until it's actually a subset of the popular language Python.

By deferring syntactic complexity, Hedy can show learners abstract concepts early. By the time that Hedy introduces random lists as early as level three, a learner of a traditional language may still be trying to remember the rules on how to print a simple message. This lets Hedy learners make interesting programs right from the start. For example, random lists can be used to make programs that tell a different story every time it is run.

2.2 Programming knowledge and misconceptions

At the same time, learning about abstract concepts opens the door to misconceptions. Programming requires different kinds of knowledge. We have seen syntactical knowledge, that is the rules of how programs should be written. Then there is conceptual knowledge, which Qian and Lehman define as knowing "how programming constructs and principles work, and what happens inside the computer"[5]. For example, knowing what happens when you assign a value to a variable, or why quotation marks are needed around strings, i.e. messages. When a learner has a mistake in their conceptual understanding, we call this a misconception. For example, a learner may mistakenly believe that variables remember all the values

that they assigned to it, when it is actually the case that variables can only hold one value at a time.

It is said that syntax mistakes are easily corrected, since they are easily identified. [paper that says this says:] That is, they do not usually pose a significant obstacle in learning to program, though this may not be true for younger learners[1]. Whatever the case, it is hard to deny that syntax errors are easily identified; a program containing a syntax error will simply complain. Misconceptions, however, are a more subtle obstacle. They are established early, and are not easily identified since they reside inside a learner's mind. Finally, they are difficult to change once established.

Both syntax and conceptual knowledge are needed to develop strategic knowledge. Strategic knowledge means knowing how to combine syntax and conceptual knowledge to write programs that solve problems that learners haven't encountered before. This is arguably the most exciting kind of knowledge for learners, since it can be applied to make all kinds of new, meaningful programs. Since misconceptions hamper conceptual knowledge, resolving them paves the way for strategic knowledge.

2.3 Mental models

Misconceptions are connected to mental models. A mental model here means knowing step by step what happens inside the computer when a program is run. Experienced programmers can often mentally execute code and produce the same output as when a computer were to run it. These programmer do not simulate the actual transistors nor calculate the zeroes and ones that a computer uses to produce the output. Instead, they rely on their mental model of a national machine. This is an abstract machine that can execute code in general terms, so not specific to a particular programming language, which can be understood by humans and used to reason about programs.

2.4 Program visualisation

Researchers have tried to resolve learners' misconceptions and improve their mental models, by visualising notional machines with interactive program visualisation tools. These visualisations often use metaphors to connect programming concepts to things that learners are already familiar with, like representing variables with labels.

2.5 Aim of the study

In short, Hedy learners could benefit from such visualisation tools to resolve misconceptions and improve their mental models. However, no such tool currently exists for Hedy code. Since Hedy introduces the concept of variables as early as level two, it is possible to write multiple variables assignments in the same program early on. This is conceptually quite complicated, and may thus lead to all sorts of misconceptions from the onset. It is the aim of the current study to help learners resolve such variable misconceptions by creating an interactive program visualisation tool for Hedy. The usefulness of such a tool for Hedy code remains to be seen, so we have arrived at our research question.

To what extent can an interactive program visualisation tool help to resolve multiple variable assignment misconceptions in young users of Hedy for level 2 example code?

3 Related Work

To build a program visualisation tool for Hedy that is effective, it is imperative to look at other such tools and their approaches. An understanding of previous work, allows us to identify potential pitfalls and apply lessons learned, and may be a source of inspiration for our tool.

3.1 Lack of engagement and learning principles

First, it is important to gain an overview of the field. In their 2013 review of programming visualisation tools, Sorva et al. propose a framework for the engagement of visualisation tools, as makers of these tools regard engagement as what makes a visualisation effective [8]. Here, engagement is the extent to which a user is engaged and able to interact with the visualisation, and the extent to which the user can control and change the program to be visualised.

Though Sorva et al. conclude that program visualisations have a positive effect on learning, they find insufficient evidence to conclude what exactly makes a program visualisation more effective than others. They note that most studied tools only have low levels of engagement, which makes it impossible to compare the relative effect of engagement.

So it is thought that engagement makes a visualisation better, yet there are not enough tools in the literature that are engaging enough to support this claim. In their 2016 Hidalgo-Céspedes et al. looked at recent program visualisations and investigated how they incorporated learning principles that[2]. Looking at sixteen learning principles based on Vygotsky theory of learning, they found support for only two of them, and almost no support for the rest. This means only two principles were consistently incorporated in recent program visualisation tools. In fact, for eight principles there was no support at all, meaning that none of the tools they looked at incorporated these principles. Furthermore, they recommend to

incorporate these principles via gamification and visual concrete metaphors.

So most existing tools are not fully engaging, nor do they incorporate learning principles, which may limit their effectiveness. It is clear then that we must consider both engagement and learning principles in the design of our tool. This puts us in a better position to answer our central question, as we can be more confident that our tool approaches the extent to which an interactive program visualisation tool can resolve misconceptions for Hedy code.

Figure 1: Human Resource Machine

3.2 Other tools

We have looked at the landscape and arrived at two requirements for our tool: it must be engaging, and it must incorporate learning principles. As to how this can be accomplished, we turn to Hidalgo-Céspedes et al.: visual concrete metaphors and gamification. For inspiration, we will turn to other programming visualisations utilising metaphors and gamification.

It may be surprising that good examples of programming visualisations that make use of gamification and concrete visual metaphors, do not come from the research community. Instead, these are commercially successful games presented as puzzle games, that require the user to program in an assembly-like language. At the same time these games effectively teach the user to write assembly-like code. Programming constructs are introduced gradually; players are given multiple contexts to practice with the concepts through the use of levels; puzzles are problem-solving exercises with multiple solutions; narrative devices and game elements motivate the player to complete levels and improve their solutions.

Crucially, these games employ a cohesive analogy in their program visualisation, making use of concrete visual metaphors. For example, in Human Resource Machine (Figure 1), the player writes assembly code that a little office worker then executes. The code state is visualised inside an office. Values are little tiles that are delivered and shipped away via conveyor belts, which represent input and output. The office worker dutifully manipulates these tiles, putting them on a board on the floor, registers, and shuffles them around and combines them as stipulated by the player's program. The code itself is also visualised, with the use of arrows showing jump destinations, and connected to concrete elements inside the office, with labels inside code having the same concrete appearance as in the office. Notice how the visual metaphors are concrete, connected to concepts that players are expected to already be familiar with, such as conveyor belts, labels, and tiles that can be picked up and manipulated.

In another game called Exapunks (Figure 2), the player learns to program concurrent assembly machines, pre-

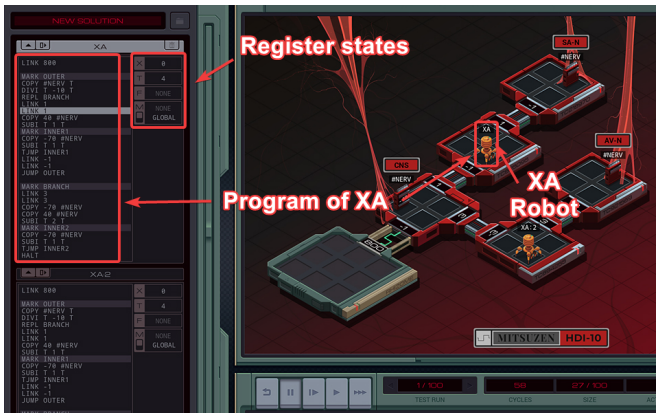

Figure 2: Exapunks

sented as a hacking puzzle game with a cyberpunk narrative, visualised by little robots that crawl inside a computer network. In contrast to Human Resource Machine, the world the robots inhabit does not visualise the program's state, but represents a separate world with its own goals. Code is confined to the memory circuits, and the robots only visualise code execution in terms of its effect on the robots.

So programs can be visualised to a greater or lesser degree, as we see in Exapunks and Human Resource Machine. Between the two it's clear that Human Resource Machine is a better visualisation tool. On the extreme end of the spectrum, there is Opus Magnum (Figure 3), where the code and the visualisation are inseparable. The visualisation encapsulates all information about program structure, execution and state, and holds even more information than the code, as the interconnection between concrete elements influences the simulation. While impressive, any resemblance to actual code is lost, and the visualisation fails to bridge known notions to programming constructs; it's no longer a metaphor. Of these three games, Human Resource Machine strikes the best balance, visualising programs with metaphors and a cohesive

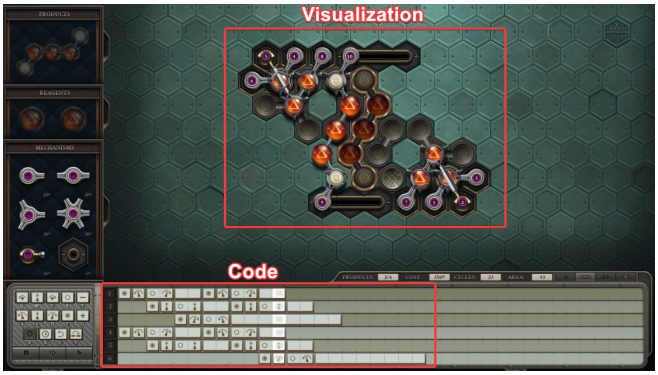

Figure 3: Opus Magnum

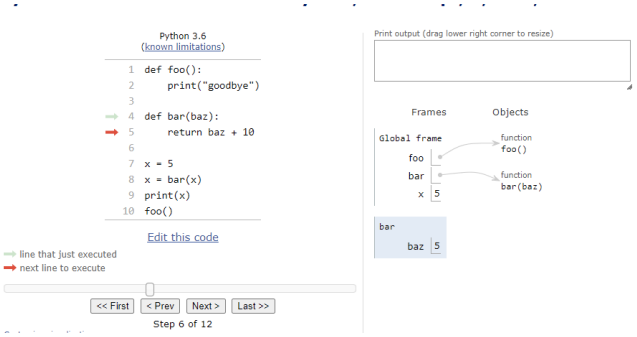

Figure 4: Pythontutor.com visualising a python program

analogy without losing the connection with the original code.

Compare these games with pythontutor (Figure 4), a popular visualisation tool for python learners, and we see a stark difference. Pythontutor exclusively portrays the visualisation with abstract metaphors; we see lists and tables connected via arrows. While the tool is immensely useful, for the user there are no elements to connect the visualisation to familiar, concrete ideas, especially for the young user who has only just started to develop their abstract reasoning[piaget].

Finally, there is Gidget [cite gidget paper] (Figure 5), a debugging game from the literature about a defective computer that requires the help of the player, meant to teach players programming. Gidget uses gamification to structure the lessons as missions and narrative to engage players. Gidget, the main character, is fallible so that players learn that computers are not all-knowing entities. Programs may contain mistakes, and the blame is shifted to Gidget's limitations rather than those of the players.

We also see concrete metaphors in the world section. Gidget is a visual character in the world. Gidget has a personality, just like anyone, and tries to interpret the code step-by-step. There are grass and brick tiles to demarcate walkable areas, and the goal tile is visualised as an animal that Gidget needs to get to.

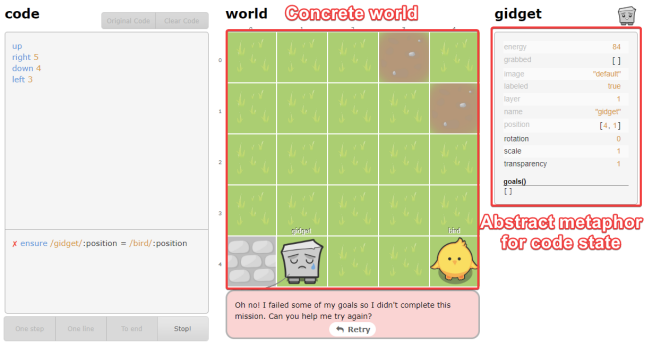

Figure 5: A mission in Gidget

However, the state of code itself is visualised abstractly, as Gidget's internal state, see Figure 5 under the right-hand panel.

So far we have seen five examples of programming visualisation tools, which use gamification and visual concrete elements to varying degrees. We must be careful not to visualise too little, nor too much, and ensure that the visualisation maintains a connection with the code. Gidget is an example of how to use gamification to structure programming lessons in an engaging way, but lacks concrete visual metaphors for its code. It seems appropriate for our study to combine elements from these examples to increase engagement and adhere to learning principles where possible, especially through gamification and visual concrete metaphors.

4 Design

From our exploration of previous work, we have learned of two key requirements that can make our tool more effective. We should make our tool engaging, and incorporate learning principles. In this section, we will outline the core ideas behind our tool that adhere to these requirements, especially via gamification and visual concrete metaphors.

4.1 Active Participation

We will first look at the idea making students an active participant in the visualisation. Most program visualisations simply carry on by themselves without requiring input from the user, or are controlled via an interface similar to a video player or a debugger. To increase engagement with our tool, we will instead require the user to find the next element of interest inside the visualisation, similar to a point-and-click puzzle game.

4.1.1 Lower Cognitive Load

The pace of our visualisation will be controlled by the user, so that only small chunks of information are imparted on

the user at a time. This decreases the likelihood that we overwhelm the user, i.e. cause cognitive overload, and we therefore give the user a better chance of really engaging with the information.

4.1.2 Hunting For the Next Element

By hiding the interactable elements within the visualisation, we ask to user to engage more directly with it, as they need to consider the spatial relationships within the visualisation to successfully find the next element. We hope that increasing the level of engagement required, by both challenging the user and by focusing their attention to the visualisation, makes them more likely to notice incompatibilities with their own misconceptions.

4.1.3 Activity-goal Association

We wish to more closely associate finding the next element with the goal of resolving misconceptions. It's been observed that people are more intrinsically motivated to engage with activities when these activities align with their goals, i.e. the reward for engaging in the activity fulfils their goal[9]. Furthermore, this motivation increases when they are rewarded earlier[9].

Users who want to find the next element, are more inclined to closely follow the visualisation, as this gives them the needed clues to find the next element. Simultaneously, this attention is rewarded in relation to the goal of resolving misconceptions, because in closely following the visualisation, they may be surprised by it and discover incompatibilities with their own ideas: misconceptions. Thus, scouring for clues and actively participating in the visualisation helps them to identify the next element, which is closely aligned to resolving misconceptions. By dividing the visualisation into smaller pieces and introducing more interaction steps, engaging with the visualisation is rewarded earlier, the student does not need to wait a long time before applying their observations to finding the next element. Most of all, we expect that hunting for the next element is fun, like a mini-game, so paying attention is also more quickly rewarded with this fun activity.

4.1.4 Integration into Hedy

That players experience heightened engagement from the activity-goal association presupposes that the user is motivated to resolve misconceptions in the first place. Though the tool may be fun to use on its own, it's ultimately a learning tool, and we expect that higher engagement makes it more effective at resolving misconceptions. Therefore, we recommend to take the approach of conceptual contraposition when integrating the tool into Hedy[2]: this means to make the learner aware that they hold a misconceptions, and then immediately offering an alternative explanation. Within Hedy, we envision a task asking students to predict the output of programs. It is shown that such tasks by themselves can improve learner's understanding[7]. When they make a mistake, they are made aware of their misconception, and are both more motivated to resolve it and more open to change their mental model[2]. This would then be a good time to offer the tool as a way for them accomplish their goal of resolving their misconception. Cues from Gidget may even be taken, by contextualising the task as a way to help a fictional character who cannot predict the output of a program. For instance, special adventures may be integrated into Hedy with a broken "Execute code" button, accompanied by a character who is in trouble because they don't know to execute the code. The student may be motivated to help this character while also resolving misconceptions for themselves.

4.2 Explicit Code-Metaphor Connection

Second is the idea of explicitly linking concrete elements to the code. Inspired by Human Resource Machine, it is our approach to make the concrete visual elements originate from the code that they are linked to. We want to make clear the connection between code and metaphor.

4.3 Metaphor-Misconception Incompatibility

Third is the use of concrete visual metaphors that are incompatible with misconceptions. Different metaphors are not equally suited to combat misconceptions. For example, the box metaphor for variables, that variables are boxes that you can put values into, does not preclude the misconception that variables can hold multiple values. The label metaphor, on the other hand, is incompatible with this misconception, because a label can only be attached to a single object. Metaphors should be used that are, as much as possible, incompatible with common misconceptions. Ideally, metaphors should interconnect to form a cohesive analogy, such as the office analogy in Human Resource Machine. Disconnected, separate metaphors may form a less compelling and convincing model, and risk being rejected by the student because they find the visualisation not intuitive.

4.4 Incorporate What Children Find Motivating

Last is to reflect what young students find motivating, with a light-hearted appearance and a fun protagonist. While pythontutor may be perceived as boring by young people, Gidget is much likely to be perceived as fun and

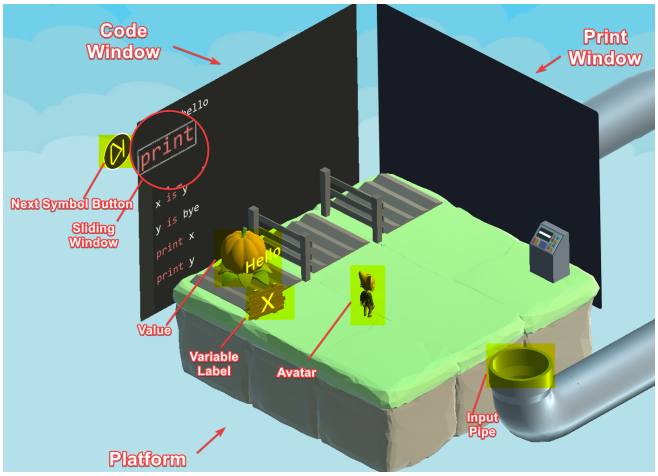

Figure 6: An overview of the tool

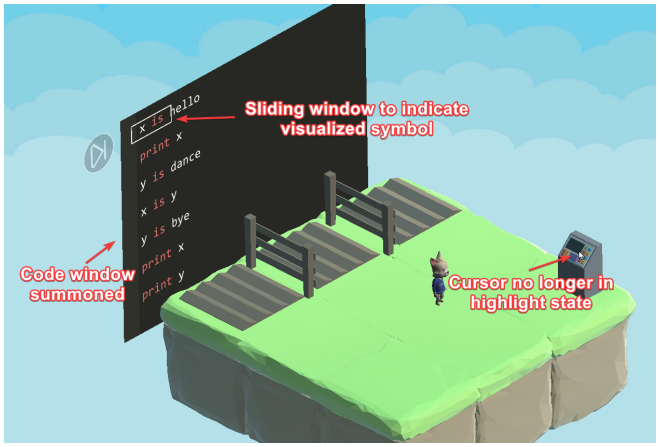

Figure 7: Code Window at the start of the game

Figure 8: Clicking on pulsating sliding window summons a new element

engaging, because Gidget's design is informed by child psychology. We expect narrative and game elements to be far more stimulating than abstract lists and arrows, and we expect an expressive and colourful presentation to better hold a young person's attention than a dull, serious presentation.

5 Implementation

What follows is an outline of the implementation of our interactive program visualisation tool. It is presented as a point-and-click game, and it incorporates the design ideas from chapter 4. It was created with the Unity game engine and C-sharp, making use of freely available models from the internet.[1] [2]

5.1 Elements of the Tool

The tool was implemented as a 3D environment, which consists of 3 main elements: the code window, the platform and the print window. Additionally, there is an avatar stood on the platform. An overview is shown in Figure 6. The goal is to click on the right objects on the scene to advance the visualisation step by step.

/subsubsectionCode window

The first main element is the window on the left hand side, in which the Hedy level 2 code appears. It is modelled after the code editing window from the Hedy website, to ensure that Hedy users will be familiar with its meaning. The window is not visible at the start of the game, and needs to be summoned by clicking on the console in the lower-right corner of the platform. This is to introduce the concept that clicking on the right object makes something happen, before linking this to program visualisation. The code presented in this summoned window will be visualised next. The pace of the visualisation is directed by the player, as they need to find the right object to click on in between visualisation steps. A small, separate sliding window moves over the code symbols to indicate which part of the code is being visualised. Whenever a code symbol generates a new element for the visualisation, it magically pops out of the symbol with a puff of smoke. This reinforces the notion that symbols in code have different functions, and that students should read code carefully from left to right, and consider how these symbols relate to how code works "on the inside".

/subsubsectionPlatform The second element is the 3D platform. This 3D space represents the "inside" of the computer. It features nine tiles, of which six look like grass and three look like exposed dirt. The three dirt tiles are a metaphor for variable addresses, associated with a gardening metaphor. The gardening metaphor connects to variable declarations, variable references being replaced

[1] Source code publicly available at https://github.com/ozbayae/InteractiveHedyMetaphors/

[2] A playable version is available at https://zsesz.itch.io/interactivehedymetaphor

Figure 9: Cursor changes to indicate an interactable object has been found

by their values and assignments. When first declaring a variable, a wooden sign with the variable's name on it is planted in front of an empty patch of dirt. When assigning this variable a value, represented by a pumpkin with the corresponding value drawn on it, it is planted with a shovel in the patch of dirt with the sign that signifies that variable. If the variable holds an old value, and the patch of dirt has therefore a old pumpkin plant, the plant is uprooted and thrown away, reinforcing the idea that variable assignment replaces the old value.

With the pumpkin planted inside the ground, the patch of dirt is watered until a pumpkin plant grows out of it. Pumpkins with the value can be harvested from this plant, and whenever a pumpkin is harvested, a new pumpkin grows back. This represents that variables appearing in expressions are replaced with copies of their value. The variables themselves, the wooden signs, once planted, are never manipulated. Only their values can be changed or copied.

5.1.1 Print Window

The third main element is the print window. The print window is modeled after its equivalent on the Hedy website, and similarly appears on the right hand side. A pipe is attached to the back of the window, and the opening at the other side appears adjacent to the platform. Here, values can be put into the pipe, causing the value to be printed on the print window, accompanied by a puff of smoke.

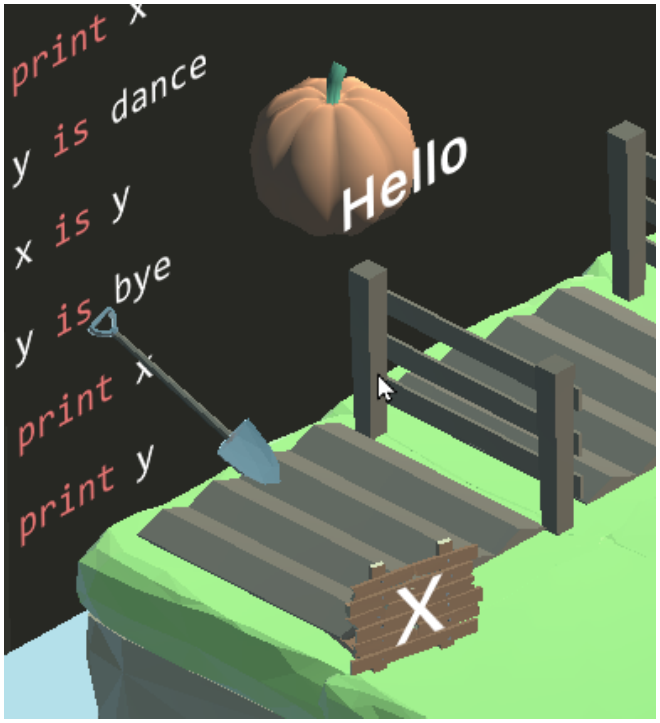

Figure 10: Pumpkin with value "hello" being planted at dirt patch of "x" with a shovel

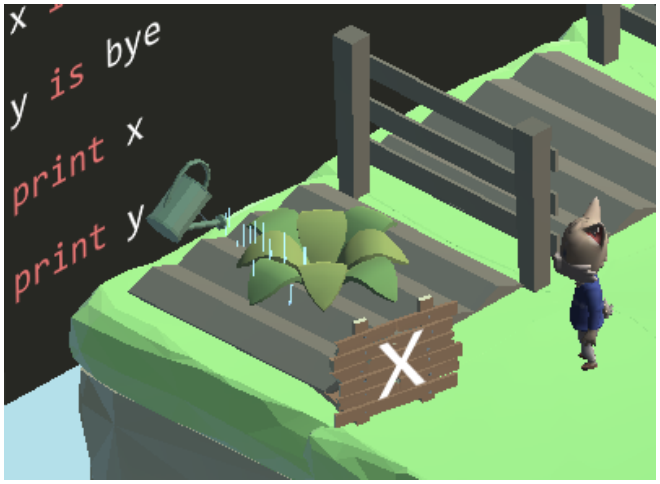

Figure 11: Watering after planting the pumpkin at the dirt patch of "x"

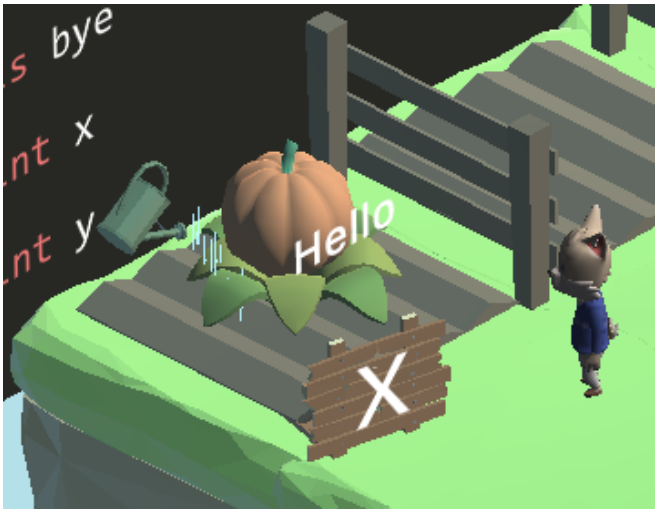

Figure 12: Watering after planting the pumpkin at the dirt patch of “x”

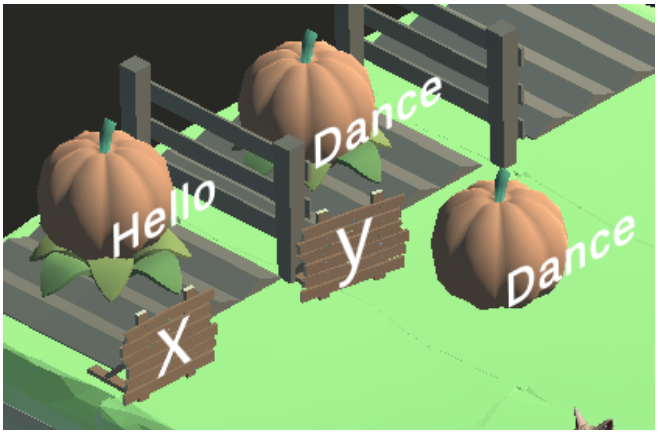

Figure 13: In “x = y”, the value of y is harvested to replace the value of x

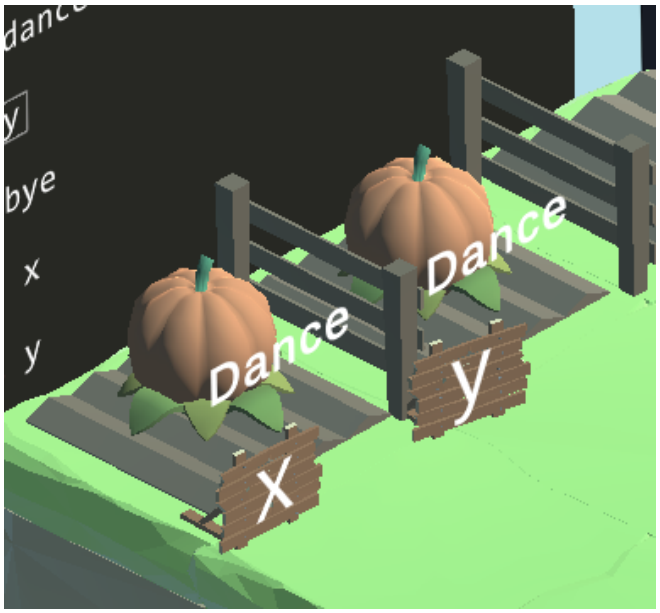

Figure 14: Situation after “x = y” has been completed

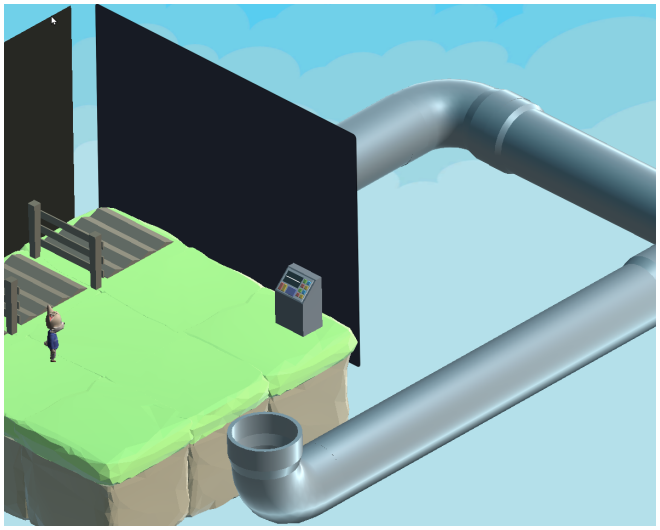

Figure 15: The print window appears on the right of the platform

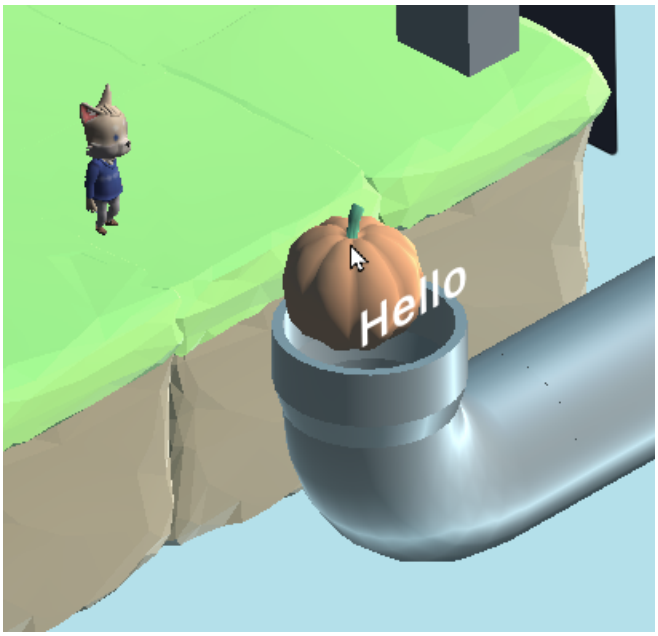

Figure 16: Hello being put into the print pipe

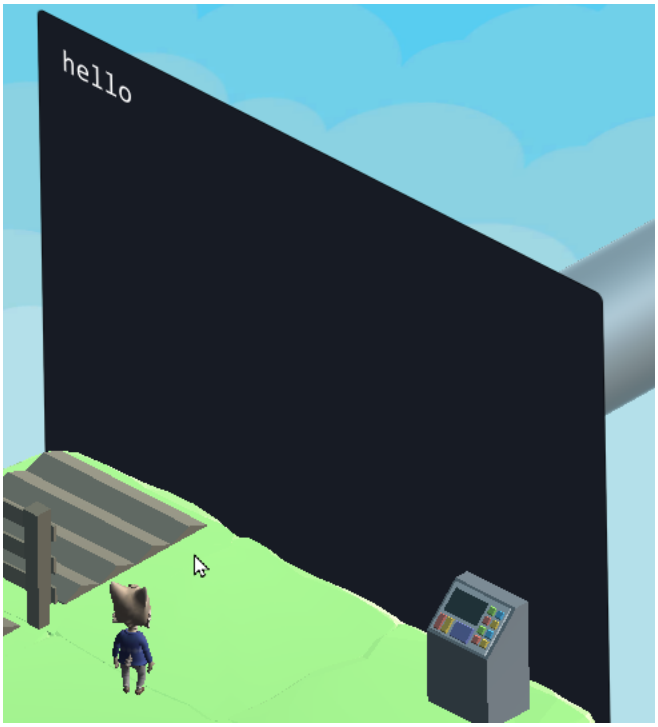

Figure 17: Print window shows “hello” after pumpkin with hello on it is put into its pipe

5.1.2 Avatar

Finally, there is the avatar, a child-friendly little fox. This avatar runs to wherever was clicked with the mouse on the platform. Currently, this is its only functionality. Though the avatar may currently imbue players with some sense of agency, and hints at personification of the computer, that it is not all-knowing but operated by a character, it is otherwise not utilised. In future iterations of the tool, the avatar may facilitate additional modes of interaction and goals for the player. For example, the player can be required to harvest pumpkins themselves, and bring them to right destination via the avatar, who laboriously hauls the pumpkin.

5.2 How it works

The goal of the player is to click on the right position on the screen, which changes each time the player has clicked on the location. Invisible objects are placed in the scene, and only one is active at a time. The mouse position to the active object is tracked; when the mouse is close enough, the cursor changes to indicate that the right position is found. These invisible objects are placed in the scene over the appropriate objects that the player should click on. For example, to advance the sliding window to the next symbol, the player should click on the “play” button in the upper left corner of the code window. When the player clicks on the right object, that interaction is deactivated, and an event is triggered to play the next scene of animations. At the end of the scene, the next interactable location is activated. These animations were keyframed by hand, and implemented via Unity Timelines, which directs a collection of animations to form a cutscene. As a consequence, the tool only works for one instance of example code. For the purposes of this study, this is sufficient. However, in real-world use, the tool should be modular so that it can accept any code, and scenes and animations should be generated by functions.

Dividing up the visualisations in smaller scenes may result in less cognitive load for the player, as the amount of new information imparted in each scene is kept small. In fact, the complete visualisation for the example program consists of thirty-six separate scenes. Furthermore, by giving the player a goal, i.e. find the next object of interest to click on, they may become more engaged. As a result, players may become more prone to discover and challenge their misconceptions. Some mechanisms that may such support this: at each point they may scrutinise the animation to figure out what the next object of interest is, taking in more information than simply glossing over the animation; they may compare their expectation for the next object of interest to the actual object of interest; the animation clips may behave differently to their expectations as they try to predict what happens.

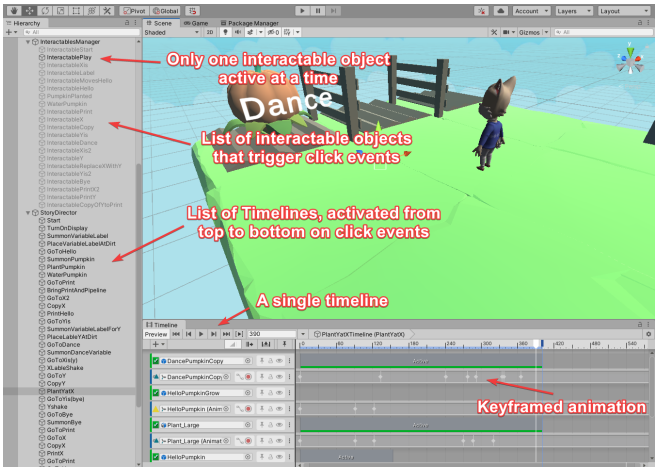

Figure 18: Tool opened inside the Unity game engine

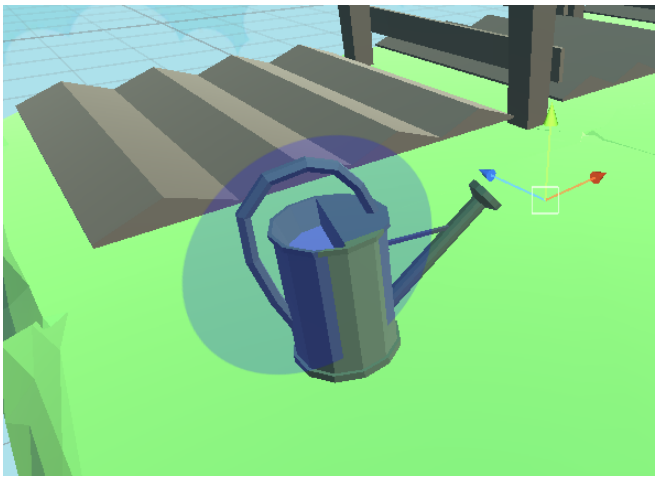

Figure 19: Invisible interactable object that triggers click event

When the program has been fully visualised, a victory screen informs the player they have finished the game.

In order to support web browsers, which Hedy runs on, the game was compiled with Unity's WebGL implementation. Thus, the tool may be embedded in the Hedy website. Support and performance for WebGL varies, but most modern browsers should perform adequately.

6 Method

In order to examine the extent that the tool described in Section 5 can resolve misconceptions, we have taken the tool to a primary school in the Netherlands. Here, we have tested students on their ability to determine the output of Hedy level 2 programs before and after using the tool. Finally, we asked them to fill out a questionnaire about the tool.

7 Participants

The participants were 8 primary school students at a public Dutch primary school near a major city in South-Holland. The school has a digital curriculum, which includes computational thinking and IT-skills, and this curriculum starts in the first grade. The 8 participants had had classes with Hedy up to level 2, which were last given at the end of 2021. The participants were equal part female and male.

8 The course of the experiment

First we asked parents for informed consent with form, that explained the nature of the study, and how the data of participants would be handled. At the school at the "groep 8" classroom, I introduced myself, gave a short pitch about what the experiment would entail, including they would get to play a game about Hedy. It was stressed that it was voluntary, and that I would ask for their permission formally later. It was possible to conduct the experiment in a different, empty class room, only the participants would be present. There were 8 participants. We invited 4 of them to the room first, asking them to bring their school-provided laptops. When they were finished, we invited the last 4.

I explained what they were expected to do. First, to sign a form that explained some things about the study, including how their data would be handled, if they agreed to participate. Next, to answer some test questions that tested their code comprehension of a couple of example programs from Hedy level 2. They were told that their answers were anonymous.

In the test, they were shown a Hedy program inside the Hedy code editor. They were then asked to write down what output they thought would be displayed in the print window, after running the program in Hedy. They were also asked to share why they thought the program would produce that output. All this was asked for three programs. In Figures 20, 21 and 22 the three test questions are shown along with the correct answer.

The reason that this approach was chosen over a qualitative interview, was to ensure that the interviewer did not bias the participants' answers. Furthermore, this approach mirrors the programming activity inside Hedy that is intended to be paired with the tool.

The children were then asked to play the tool in pairs. The reason to pair them was to simplify the logistical problem of having enough laptops with the tool available, and that pairs of children more readily share their thoughts. These thoughts were valuable, as were how they interacted with the tool. Therefore, the computer screens were recorded for two of the pairs, and the audio

```
fruit is banaan
fruit is appel
print fruit
print fruit
```

```
appel
appel
```

Figure 20: First test question, correct answer below

```
voornaam is hedy
achternaam is code
voornaam is achternaam
achternaam is voornaam
print voornaam
print achternaam
```

```
code
code
```

Figure 21: Second test question, correct answer below

```
kleur1 is rood
print kleur1
kleur2 is groen
print kleur2
kleur1 is kleur2
print kleur1
print kleur2
```

```
rood
groen
groen
groen
```

Figure 22: Third and last test question, correct answer below

of the play through was recorded for every pair for later analysis.

The participants were encouraged to think out loud, and to take turns in playing the tool. I briefly explained the goal of the game: to find and click on the correct place in the screen to advance the game, until the game was complete. I left them to complete the game otherwise.

After completing the game, I asked them to retake the test, and that it was possible that their answers may have changed after playing the game. See the appendix for this version of the document. The test was scored as follows: each answer is awarded two points for a correct answer, one point for a partially correct answer, and zero points for an incorrect answer. With three questions, there was a maximum of six points. Blank space accompanied each test question for participants to indicate their motivation behind the answer. This is to gain insight into participants thought process, and allows us to identify misconceptions in their thinking.

Afterwards, we asked them to fill in a usability questionnaire. We stressed that this was anonymous. We opted to not conduct interviews, because we had only potential interviewer, who was creator of the tool, which could be reason for the participants to answer in socially acceptable way, obscuring the truth. In contrast, an anonymous questionnaire on paper places distance between them and the social context of the study. The participant could rate each statement on the questionnaire with values ranging from 5 (= completely agree) to 1 (= neutral). Every

statement was phrased positively, so that higher average of ratings indicated an overall more positive opinion. Though questions mostly pertained to the tool, some statements were about Hedy, so that we may compare the two. There was also the opportunity to fill in open feedback, so that participants could share valuable feedback we could not account for.

Finally, the questionnaires and the answers to the tests were associated via numbers. Each participant received a set of three documents that had the same number. This way, participants remain anonymous but answers across the documents by the same participant can still be retrieved. This is useful for comparing answers before and after the tool per participant.

Table 1: Participant's answers in the questionnaire

Participants' ratings –>	1	2	3	4	5	6	7	8	Average rating per question
Questions (below)									
Part 1									
I think that the game is a fun way to learn to understand code.	4	5	5	4	2	4	4	4	4
I think that the Hedy-website is a fun way to learn to understand code.	4	4	4	3	3	3	4	4	3.625
I feel at ease when I'm learning to understand code with the game.	3	5	5	4	4	4	4	4	4.125
I feel at ease when I'm learning to understand code with the Hedy-website.	4	4	4	3	4	4	4	4	3.875
I would like to play the game again.	3	5	5	2	4	5	2	4	3.75
I would like to use the game to continue to learn to understand code better.	5	3	5	2.5	1	4	3	3	3.3125
I think that the game can help me (better) understand how Hedy-code works.	5	4	4	2	4	5	3	3	3.75
I think that the game is a useful way to become excited to learn to better understand code.	4	5	5	3	4	4	4	4	4.125
I would recommend the game to someone I know (friends, siblings)	4	5	4	3	3.5	5	1	2	3.4375
Part 2									
I think that the game is easy to use.	5	5	5	3	4	4	4	4	4.25
I think it's easy to learn to understand code with the game.	5	5	5	3	1	4	3	3	3.625
I knew what I was supposed to do during the game.	3	5	4	3	4	2	3	3	3.375
I think that the game clarified the code.	5	5	5	3	2	4	3	3	3.75
I am sure that I can learn to understand code with the game on my own, even without assistance.	4	4	4	1	1	4	2	2	2.75
I am sure that I can learn how Hedy-code works with the Hedy-website.	3	3	3	3	3	3	3	3	3
Part 3									
I want the game to be added to the Hedy-website.	5	5	4	4	4	5	3	4	4.25
I would use Hedy more often, if the game was added to to the website.	4	5	4	3	4	4	2	3	3.625
I think that there is enough explanation about code on the Hedy-website.	5	3	3	3	3	3	3	4	3.375
I want to be able to play the game with my own code.	2	4	4	3	4	4	4	4	3.625

I think that I would better understand Hedy code, if someone explained it to me.	3	2	3	5	5	3	5	5	3.875
I think that I would better understand Hedy code, if someone explained it to me by showing the game.	3	5	5	4	1	3	4	5	3.75
I found the game to be interesting.	4	5	5	4	4	5	3	4	4.25

Table 2: Participants' answers to test questions

Participant	Question 1 (before)	Question 2 (before)	Question 3 (before)	Question 1 (after)	Question 2 (after)	Question 3 (after)	Score (before)	Score (after)
1	banaan appel	hedy code	rood groen	appel appel	hedy hedy	rood groen rood rood	0	4
2	er komt appel of banaan op elke regel ...	code hedy	rood groen groen rood	[leeg]	code hedy	rood groen groen groen	0	2
3	banaan appel	code hedy	rood groen groen groen	banaan appel banaan appel	code hedy	rood groen groen groen	2	2
4	banaan appel	[crossed out]	[leeg]	banaan appel banaan appel	hedy code	rood groen rood groen	0	0
5	appel banaan	hedy code	groen groen	banaan appel	code code	rood groen groen groen	0	4
6	fruit fruit	hedycode	rood groen groen groen	banaan banaan	code hedy	rood groen groen groen	2	3
7	[empty]	[empty]	[empty]	Er komt fruit	Er komt Hedy code te staan	[leeg]	0	0
8	[empty]	[empty]	[empty]	Fruit er staat dan fruit	Hedy code Er komt Hedy code te staan	er komt rood en groen te staan.	0	0

9 Results

In this section, we present the data that was collected during the experiment. The data is then further examined and summarised. Table 2 shows the answers given by participants before and after having used the tool, as well as their score. The results of the questionnaire are shown in table 1.

9.1 Test Scores

A preliminary look at the test scores before and after having played the tool once, shows that four of the eight participants improved their score. These are participants 1 and 2, and 5 and 6. It's worth noting that these participants were paired: 1 and 2 played the tool together, taking turns, as did 5 and 6. Participant 1 and 5 showed the most improvement: both increased their score from 0 to 4, while participant 6 improved their score by only 1 point, from 2 to 3. The other four did not improve their scores. In fact, they all maintained a score of 0, except for participant 3, who maintained their initial score of 2.

9.2 Questions

When looking at the questions, we discover that performance was not equal on every tested program. For the last question, two of the eight participants already scored full points before the playing the tool. After playing the tool, these participants did not change their correct answer, 2 other participants also got full points for that question, and 1 participant got it partially correct. That means most participants answered this question correctly. Perhaps this should not come as a surprise: the structure of the program in the last question is very similar to the visualised program.

While the scores are a convenient shorthand, they don't tell the full story. For example, participant did not improve in their score, and did not fill in anything for the last question in the first round. After having played the tool, however, they filled in an answer that was very close to the correct answer, missing out on partial credit by a narrow margin.

Furthermore, not all questions seem equally difficult. While 2 participant got the last questions right before the tool, no one got the first two questions right initially. After the tool, only 1 participant got full credit for the first question, and 1 other got partial credit. Similarly for question 2, only participant 5 got that question completely right, and participant 1 got partial credit.

The inequality of questions should be kept in mind when comparing test scores, as the scores do not tell how participants answered, and does not account for the higher difficulty of the first two questions over the last question.

9.3 Misconceptions

We have looked at the improvement of overall scores of participants, and compared the questions and the performance differences on these questions. Finally, we will scrutinise the contents of the student's answers for misconceptions.

9.3.1 Test Question 1

A recurring misconception in the first question, shown in Figure 20, is that variables can hold multiple values. This is made evident in the answers of participants 1, 2, 3 and 5. Before the tool, each participant answered that the consecutive print statements would print something different, even though the same variable is printed. Furthermore, each participant motivated their answer by pointing out that the variable was both the value in the first assignment and the value in the second assignment. Uniquely, participant 2 thought that there was a 50 percent chance that each print statement would either print "banaan" or "appel", enumerating the four possible combinations of output.

Another misconception for this question was possibly held by participant 6. They answered "fruit, fruit", stating that fruit is what is printed. It's possible that they believed that variables are not replaced with their value, because print interpreted "fruit" not as the variable, but as the word to print. This misconception is expected, as quotation marks to differentiate between these cases are not introduced until later levels.

Participant 1 seems to have abandoned this misconceptions after playing the tool, giving the correct answer and stating that fruit is "banaan" at first, but then turns into "appel".

9.3.2 Test Question 2

A common misconception for the second question, shown in Figure 21 was that the third and fourth statements, switched the values of the variables around. It may be the case that the naming of the variables induced this misconception. "Voornaam", meaning first name, and "achternaam", meaning surname implies direction in these variables, and the participants may have had the expectation that the program ought to switch these names around. The alternative, being that the names are not switched around, would mean that the program is rather pointless, which may be cause to abandon that idea. Furthermore, since each variable name implies what should be stored in it, this may also induce the misconception that natural language meanings of variable names has an effect on the program. By naming the variables first name and surname, this implies that their values are a first name and a surname, and people do not often have the same

first name as surname. Therefore, to the participants it may have seemed unlikely that the variables would end up holding the same value.

Participant 1 and 5 seem to have resolved this misconception after playing the tool, motivating their answer by stating that "voornaam" became "achternaam". Partipant 1 did however answer "hedy hedy", getting the order wrong.

9.3.3 Test Question 3

As for the last question, shown in Figure 22, most participants had little to say about their answers before the tool, so no misconceptions could be identified with any certainty. However, more participants were able to motivate their answer after playing the tool. Again, participants struggled to motivate their answers, but those who got it right, could explain that while the variables first differed in value, one was later assigned the value of the other.

9.4 Comparing Hedy and the tool

Some analysis of the questionnaire data yields some interesting results. First, the average rating for positive statements about the tool was 3.75, yet the average for Hedy was 3.46. While interesting, these numbers should not be compared: there were more statements about the tool than about Hedy, and there was an additional statement for Hedy that was not applicable to the tool.

However, when computing the average of only the comparative statements, i.e. statements that asserted the same about either Hedy or the tool, we find an average assessment of 3.625 for the tool, and 3.5 for Hedy.

The average rating of every statement was 3.7, providing a baseline to compare deviations with. The only statement uniquely about Hedy, "I think that there is enough explanation about code on the Hedy-website", deviated a bit from the overall average with an average rating of 3.375.

9.5 Categories

When looking at commonalities between individual participants, we find 4 categories. The first are those who showed no improvement in the test, but rated the tool highly nonetheless. Participant 3, for example, evaluated the game overall with 4.58, much higher than the average, even though they did not improve in the test. They indicated that they thought understood the code better because of the game, though misconceptions remained present in their test motivation, e.g. the misconception that variables can hold multiple values.

The second category are those who showed improvement in the test, and rated the tool highly as well. For example, participant 1 did resolve the misconception that variables can hold multiple values, as indicated in their test answers and provided motivation. In fact, they went from 0 out of 6 points to 4 out of 6. They rated the game on average with a 4. Participant 2 went from test score 0 to 2, and rated the tool on average with a 4.71 (their average rating of Hedy was 3.5).

The third category are those who similarly showed improvement in the test, yet rated the tool not as highly as the second category. Participant 5 in particular rated the game on average with a 3.03, and rated the game 2.33 when compared to Hedy, which they rated with 3.33. This was the lowest rating of the tool of all participants in the comparative questions. Surprisingly, this participant went from a 0 in the test to a 4, and they were the only one to get the second test question right, after having played the game.

The final category did not improve in the test, and rated the game similarly to category three. Though participants 7 and 8 did not fill in any test question before the game but did after, whether this can be attributed to social pressure to answer or raised confidence in their understanding remains unclear. Together with category three, they indicated that they would have liked more explanation accompanied with the game, and were initially confused by it.

9.6 Highest and lowest rated statements

Some statements that were rated higher than average, were "I think that the game is easy to use", with an average rating of 4.25, as were "I want the game to be added to the Hedy website" and "I found the game to be interesting". "I feel at ease when I'm learning to understand code with the game" (Hedy's counterpart was rated 3.88) and "I think that the game is a useful way to become excited to learn to better understand code" were both rated with a 4.13. Finally, "I think that the game is a fun way to learn to understand code" was rated with a 4. This was a higher rating than Hedy's counterpart of this statement, which was rated with a 3.63.

The three lowest average ratings for the tool were 2.75, 3.31 and 3.38, for the following questions respectively: "I am sure that I can learn to understand code with the game on my own", "I would like to use the game to continue to learn to understand code better", and "I knew what I was supposed to do during the game".

10 Discussion

In order to learn to program more meaningfully, students must grow their programming knowledge beyond the syntactical. However, misconceptions that may be present students, can significantly impair their conceptual understanding of code. Simply writing more code may not be

sufficient to resolve such misconceptions. Instead, special learning experiences may be needed [source]. Yet, at the time of writing, Hedy does not currently offer special facilities for students to resolve misconceptions about Hedy code. Program visualisations may be an effective tool to resolve such misconceptions in children. Yet, no such tool yet exists for Hedy-code.

It was the aim of the current study to investigate whether an interactive program visualisation tool for Hedy example code is an effective approach for resolving misconceptions in young Hedy users. Restricting our scope to the initial two levels of Hedy, the study sought to answer the question: "To what extent can an interactive program visualisation tool help to resolve multiple variable assignment misconceptions in young users of Hedy for level 2 example code?"

Our investigation led to the creation of a tool of a game and program visualisation tool for Level 2 Hedy code. Eight children from a Dutch primary school participated in the study. They had some experience with Hedy Level 1 and 2, and were asked to play the tool. Additionally, each was tested on their conceptual understanding of Level 2 Hedy programs with multiple variable assignments, before and after playing the tool exactly once.

10.1 Improved Test Scores

The results of the tests indicate that every student started out with misconceptions. Yet four of the eight participants improved in the test. As students were given minimal guidance, and only played the tool once, this provides evidence that the tool helped these four students to resolve misconceptions they had about multiple variable assignments.

10.2 Effects of type of misconceptions on scores

Even though this group showed improved scores, no one attained a perfect score. This may mean that the tool is better at resolving some misconceptions than other, as each test question differs. Notably, the first and second test question were each only answered correctly once after the tool. This may be due to the natural language misconception [quote the misconceptions outlined paper], that the natural language meaning of names of variables influence their values. In the example code that was visualised in the tool, variable names were "x" and "y". Since such names carry no natural language meaning, the misconception may not be challenged, as it does not interfere the student's understanding of the example. Thus, this understanding is never contradicted, and the user is not given the opportunity to discover and reject it. This indicates that the tool, in its current state, may only help resolve a limited set of multiple variable assignment misconceptions, namely those that the tool's example challenges. However, this does not indicate that the approach is limited, as it is easy to modify such a tool to include examples that do challenge specific misconceptions. Moreover, participants performed best on the last question, which was most similar to the example in the tool. This both corroborates the finding that the tool resolves some misconceptions better than others, and also hints that students may not transfer their insights gained from the tool as readily to dissimilar code. This cannot be so easily remedied, though it is plausible that playing the game with different examples improves transfer.

Our tool shows promise for such tools to resolve misconceptions in young Hedy students , as it has shown to help in resolving multiple variable misconceptions for four of our eight participants. At the same time, the tool did not manage to resolve every misconception that students had, specifically those that did not appear in the visualised program. While this may limit the extent to which our current tool can help in resolving misconceptions, as this issue may be more or less easily remedied, care should be taken to write off the overall the approach of interactive program visualisation for Hedy.

10.3 Mismatch Between Score and Attitude

Not all participants who improved, felt that the tool helped them. This is likely due to a lack of feedback. Participants were never told when they got the answer right. However, this also reinforces the need to present learning experiences appropriately. Students may miss out on valuable learning experiences, because they perceive them negatively. Our tool was perceived as engaging and fun, but to a lesser extent educational. Therefore, it is important to provide students with feedback to their answers when the tool is integrated in Hedy (accompanied by output prediction exercises similar to those used in the study). (integration not yet explained in design section, should it be? If not, remove reference to Hedy integration) [Reference to useful activities that are not perceived as fun or worthwhile]

10.4 Possible Causes of Lack of Improvement

While four students did improve, four students did not improve their performance on the test. One possible explanation is this is due to the design of the experiment, e.g. participants having only allocated little time with the tool, or due to lack of guidance during the experiment. With real world use, it is possible that that the tool proves a greater aide than what the experiment has been able

to show.

A second explanation is that the tool may be underdeveloped. In this case, it is expected that adding additional features, such as the ability to visualise one's own code, sound effects and multiple code examples, will improve the tool's utility in resolving misconceptions. Keeping this explanation in mind, the results may indicate only a lower-bound for the extent to which such a tool can resolve misconceptions in young student.

Alternatively, it may be the case that the approach of interactive program visualisation is not equally fruitful for every type of student. This may put an upper-bound to the extent that the tool can resolve misconceptions. While some students may find that the tool helps them resolve misconceptions, even when it is underdeveloped, others may find little help in the tool, no matter how refined the tool. In our results we have seen a category of participant who did not rate the tool highly, who also did not improve in the test. It is this category that we suspect may least benefit from our approach.

10.5 Extent of Resolving Misconceptions

In short, the tool brings to light a potent approach to resolving multiple variable assignment misconceptions. Our tool worked better for some students than others, and may help to resolve some misconceptions better than others. What this means for the limit to the extent that such a tool can help resolve misconceptions, depends on the explanation for why our tool performed variably. It may be the case that the tool was underdeveloped, or that it would perform better in a real-world scenario. In this case, we can only establish a lower-bound for this extent, namely the extent we have found for our tool. Alternatively, the tool may not suit some students, in which case the extent to which such tools may resolve misconceptions depends on the student. In failing to help resolve misconceptions for some participants, we may have found an upper-bound on the extent to which such a tool may help resolve misconceptions for a group of students.

10.6 Reception

The tool also serves as a proof-of-concept. It shows that a fun, engaging and colourful program visualisation, that utilises various design principles, such as from game design, can be an effective tool to resolve misconceptions. However effective such a tool may be, it would still be rendered useless if students did not like it and would not use it. Our participants, however, were enthusiastic about the tool, thought it was fun and interesting, yet helpful, and indicated that they would like to see it in Hedy. At the same time, they expressed to a lesser degree that they thought Hedy was sufficient in helping them understand code, and were more hesitant to agree that enough explanation was available on Hedy. Thus, we showed that not only the tool may be effective in resolving misconceptions, but also that Hedy students may be happy to use it to resolve misconceptions, and welcome it to Hedy.

10.7 Future Work

As it stands, Hedy does not currently have a program visualisation tool embedded in the website. This study showed that an interactive program visualisation tool can be effective in helping resolve misconceptions. Furthermore, Hedy students may welcome it as an addition to the Hedy website. Therefore, we recommend that it be added to Hedy. However, the current tool only works for one example. Before Hedy and its students can take full advantage of the tool, it first needs to be made to work for arbitrary Hedy code. In order for this to work, animations must be generated with algorithms, instead of keyframed by hand. A parser should analyze Hedy programs and add the appropriate animations to the queue, and populate the 3D scene with the right objects.

Making the tool work for arbitrary examples presents a number of other challenges. First, Hedy code with bugs lead to various exceptions, and one would expect that a visualiser would handle these appropriately. However, this greatly adds to the technical challenge, as new animations and behaviour need to be programmed to visualise programming errors. Another challenge is the changing nature of Hedy. Other, similar gradual learning approaches, only add new functionality at each step. Hedy, in contrast, also changes the syntax. If the visualiser is to work for many or all Hedy's levels, the changing syntax should be accounted for. Finally, making the visualiser work for arbitrary Hedy code, opens up many possibilities for bugs to occur. This necessitates more rigorous debugging than was needed for the tool, for which only a single scenario needed to be debugged.

A second recommendation is to investigate how the limitations seen in the study can be overcome. Students who benefited from the tool, did not necessarily perceive it as useful. For instance, we recommend that Hedy integrates the visualiser with program tracing activities on the website, similar to the test questions in the study. It is paramount to allow students to become aware of their development, as this is a source of motivation, and increases the likelihood that they continue to use the tool. Moreover, students who were confused by the tool expressed a wish for more explanation. Another way to integrate the visualiser with Hedy, is to pair it with explanation of how to use it, and information about the concepts it tries to clarify, like Gidget's dictionary.

10.8 Limitations

The first limitation of the study is that no control group was present in the experiment. A control group that completed the same test without playing the tool could have shown similar improvements to our participants. Perhaps because the time between the tests was sufficient for participants to recollect their knowledge, or because attempting the test questions was sufficiently informative on its own.

Second, the number of students who participated in the study was quite small, and were from the same class and school. This makes it difficult to generalise our results to Hedy students at large.

There was also no longitudinal data, so it's difficult to say whether our participants' engagement with the tool would have diminished over time.

Lastly, we made certain trade-offs. No qualitative interviews were conducted in favour of an anonymous test and questionnaire. This may have prevented bias, and allowed us to ask more questions in the time frame we were allocated, as we had only one interviewer, while our participants could fill in the tests and questionnaire in parallel. Still, as a consequence of this trade-off, we have less in-depth data on the specific contents of our participant's opinions and thought processes. The second trade-off was that of presenting the same test questions on the first and the second test. While we can with greater certainty say that variation in answers were due to playing the tool, it may have made it more likely that participants simply repeated their old answers, and remembered their misconception over the new insights they may have gained. This means that we may have seen more improvements if the questions differed between the two tests, at the expense of certainty of its cause.

10.9 Implications

The current study presented an interactive program visualisation tool for level two Hedy code. The results indicate that this tool can help resolve multiple variable assignment misconceptions for young Hedy students. Furthermore, we have found that such a tool may be a welcome addition to Hedy. Therefore, this study supports the further development of the tool and its use in Hedy.

Misconceptions are an important challenge for students to address. They may be present as early as level two, as we have seen in our participants. These misconceptions can remain undetected and are hard to change, and limit the student's ability to reason about and write programs. In short, they they are an obstacle for learning to program. If they are not addressed, they may continue to hamper Hedy's effectiveness as a learning system. As Hedy is growing more popular, so does its impact. A failure to address misconceptions may lead to more and more students holding misconceptions.

Our tool represents a significant step in addressing misconceptions for Hedy. However, much remains to be done. Program visualisation may not be equally effective at resolving misconceptions for every student, and there is still the issue of identifying misconceptions. This last issue is especially important, because misconceptions may be more widespread than realised. Furthermore, programming requires different kinds of knowledge. For Hedy to become a more complete learning system, it must consider not only how to address misconceptions, but also how to impart conceptual and strategic knowledge.

Programming is becoming increasingly important. At no point in history has computing technology had such a profound impact as it has today. Our youngest generation will not remember a time before ever present smartphones, as powerful as supercomputers from only decades ago. The internet, a global network of near instant information exchange that was once the stuff of science-fiction, is now ingrained in our culture, especially young culture. While this brings exciting opportunities, it is an increasing concern that computing technology is controlling our lives, especially our youngest. It is paramount to provide our children with programming education that empowers them to take control over computing technology, as it will one day fall to them to shape the future. Hedy is at the frontier of programming education, in position to prepare young people for a future of programming, so we should strive to improve it. It is the author's hope that the current study contributes to a better Hedy.

References

[1] Felienne Hermans. Hedy: a gradual language for programming education. In *Proceedings of the 2020 ACM conference on international computing education research*, pages 259–270, 2020.

[2] Jeisson Hidalgo-Céspedes, Gabriela Marín-Raventós, and Vladimir Lara-Villagrán. Learning principles in program visualizations: a systematic literature review. In *2016 IEEE frontiers in education conference (FIE)*, pages 1–9. IEEE, 2016.

[3] Benjamin Mako Hill and Andrés Monroy-Hernández. A longitudinal dataset of five years of public activity in the scratch online community. *Scientific data*, 4(1):1–14, 2017.

[4] Yu-Chang Hsu, Natalie Roote Irie, and Yu-Hui Ching. Computational thinking educational policy initiatives (ctepi) across the globe. *TechTrends*, 63(3):260–270, 2019.

[5] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24, 2017.

[6] José-Manuel Sáez-López, Marcos Román-González, and Esteban Vázquez-Cano. Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. *Computers & Education*, 97:129–141, 2016.

[7] Amal A Shargabi, Syed Ahmad Aljunid, Muthukkaruppan Annamalai, and Abdullah Mohd Zin. Performing tasks can improve program comprehension mental model of novice developers: An empirical approach. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 263–273, 2020.

[8] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, 13(4):1–64, 2013.

[9] Kaitlin Woolley and Ayelet Fishbach. It's about time: Earlier rewards increase intrinsic motivation. *Journal of personality and social psychology*, 114(6):877, 2018.