Aufgabe 1

• Which number and forms of subways, number of exits, number and strategies of mice, playing time, mechanism of controlling cats, etc., provides the most exciting gaming experience?
Which variations did you try out?
1. Versuch:
30sec Spielzeit; 2-5 subways; 2-5 exits; 10 fast mice moving every milliseconds
Ergebnis: nicht spielbar (Mäuse sind zu schnell)

Weiterer Versuch: …

• Which playing strategy of cats is most promising (cooperation or playing against each other)?
PLAYING AGAINST EACH OTHER

Aufgabe 2

• Do you prefer to program in a dynamic or static language?
Why?
dynamic language = Python, Smalltalk
rapid prototyping, scripting, ability to adapt and iterate quickly

static language = Java
early error detection

Own opinion: I like the hybrid one :D like TypeScript, offer a hybrid approach by adding optional static typing to a dynamically typed language, providing benefits of both worlds.

• How important is the run-time penalty of an interpreted language like Smalltalk?
Interpreted languages, including Smalltalk, typically have slower execution speeds compared to compiled languages. This is because an interpreter reads and executes the source code directly, translating it into machine code instructions on the fly. In contrast, compiled languages translate the source code into machine code ahead of time, which can lead to faster execution.

However, it's important to consider that the relative performance of programming languages is constantly evolving. Advances in interpreter technology, just-in-time (JIT) compilation, and optimization techniques can narrow the performance gap between interpreted and compiled languages. Additionally, the increasing power of

modern hardware can also mitigate the impact of slower execution speeds in many cases.

Own opinion: mir ist das nicht wirklich aufgefallen, außer wenn man console output macht (Transcript show:) z.B. in einer loop

• Is the concept of “personal computing” (each person has its own customized system) still adequate? Are there any good alternatives?

Personalized Development Environment: Smalltalk and Squeak provide integrated development environments (IDEs) that offer a high level of customization and personalization. These IDEs allow developers to tailor the development environment to their preferences, such as layout, toolbars, code editors, and debugging tools. This level of customization enhances the personal computing experience by enabling developers to create a development environment that suits their specific needs and workflows.

- designed for use by only one person at a time
- für 1Personen Projekte sehr gut, zum scripten
- für Zusammenarbeiten an Projekten ist das nicht sinnvoll, heutzutage in Softwarefirmen müssen mehrere Leute oft gleichzeitig im gleichen Projekt arbeiten und programmieren.
- Alternative: Cloud? AWS

• How can team members cooperate when developing software together in Smalltalk?

Git Browser oder Monticello Browser
Wir haben Git Browser verwendet mit privatem GitHub Repo, Git History ist sehr schön, branches möglich

Image-based development: Smalltalk's development environment is image-based, meaning the entire state of the system, including code, objects, and data, can be saved and resumed later or shared.

• Which aspects of programming in Smalltalk do you like, which aspects don’t you like?

+ man kann sehr leicht und schnell im Workspace was scripten und direkt ausführen
+ openInHand, openInWorld Befehle im Zusammenhang mit Morphs testen
+ quick prototyping and experimentation

- Unübersichtlichkeit von parent, child Beziehungen und class variables (sehr versteckt)
- Fehler finden

Aufgabe 3

• How much work is it to specify useful assertions in Eiffel?
Initially it is a lot of work, but it leads to early error detection so it might be worth it…

• How important is the run-time penalty of assertion checking?
Es macht einen großen Unterschied (siehe binary vs run in EiffelStudio)

• How is it possible to specify pre-conditions that are, in some sense, in the subtype stronger than in the supertype although Eiffel does not allow us to redefine pre-conditions to become stronger? How is it possible to specify post-conditions that are, in some sense, in a subtype weaker than in the supertype although Eiffel does not allow us to redefine post-conditions to become weaker? (Yes, it is possible, although not obvious, and with an appropriate interpretation it is possible to do so without violating Design by Contract.)
STRONGER precondition and WEAKER postcondition in subclass is possible with a trick -> access a function/constant in an assertion, when override function/constant

man greift zb in einer precondition auf eine funktion zu, die einen wert zurück gibt.. wenn man dann einen untertyp erzeugt und diese funktion (welche den wert zurück gibt) überschreibt, kann man dort einen "strengeren" wert zurückgeben, damit ist die precondition wenn sie über den untertyp aufgerufen wird, strenger.

ich glaube er will hören, dass man die preconditions halt abhängig vom object state machen kann und den kann man ja beliebig verändern

• Eiffel supports co-variant input parameters. What are the advantages and disadvantages of this feature in practical programming?
man wird das sowieso nicht brechen, weil man intuitiv es dann immer richtig verwendet, also gibt es eigl keine Nachteile
Probiert es aus co-variant inputparameter zu haben und dann eine exception zu erzeugen! Da muss man schon Code schreiben, den man eigentlich intuitiv gar nicht schreiben will (Puntigams Worte)

• Which features of Eiffel would you like to see also in your favorite programming language?
no null objects
programmatically invariants (monitored at run-time for testing and quality assurance)
• Which features of Eiffel would you rather avoid to use?
co-variant input parameters

VOWI: "Empfinden Sie die Kovarianz bei Bedingungen in Eiffel als gut?" > eigene Meinung