UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

Software Engineering Praktikum - Kick Off

Moritz Bayerkuhnlein & Elaheh Hosseinkhani

April 2025

What is This Software Engineering Lab About?

According to the Modulhandbuch:

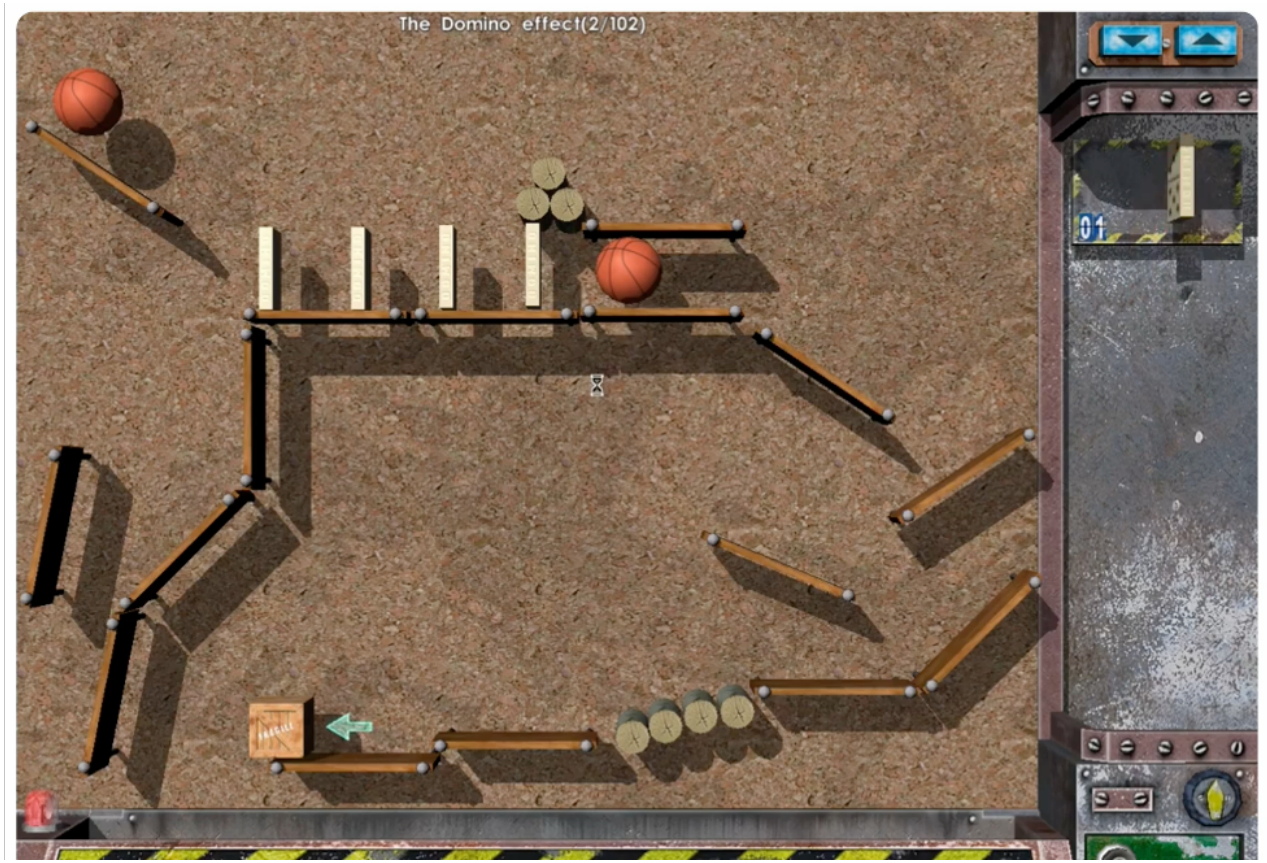
- Development of a software system from scratch
- Teamwork using project management methods
- Design, implementation, and testing

Skills You Will Gain:

- Systematic design of software systems (using OOP techniques)
- Working with UML and CASE tools
- Evaluating and improving your own software
- Presenting artifacts, following standards and meeting deadlines
- Collaborating effectively in a team and learning social skills

Important Dates

- **April 16, 2025:** Today: Kick-Off
- **May 1, 2025:** Deadline for group formation in Moodle
  Course enrollment, group selection, and GitLab project registration
- **May 18, 2025:** Deadline for registration in QIS
  Possible from May 1. The lab work counts as an exam requirement.
- **May 21, 2025:** Release I submission in GitLab
- **June 25, 2025:** Release II submission in GitLab
- **midterm meeting** Presentation of the interim results and getting feedback before the final submission
- **July 23, 2025:** Release III submission in GitLab
- **End of July:** Final presentation in the PC pool



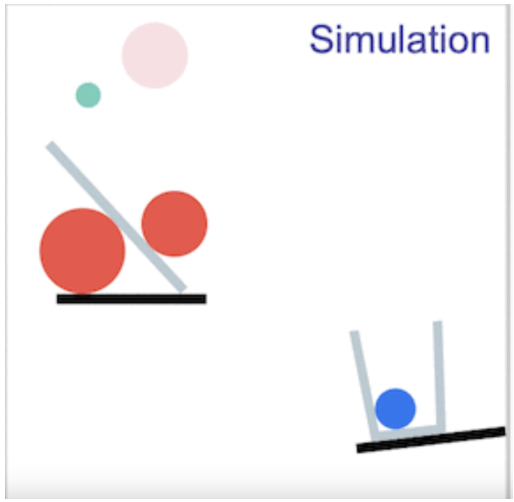
Screenshot from the original game

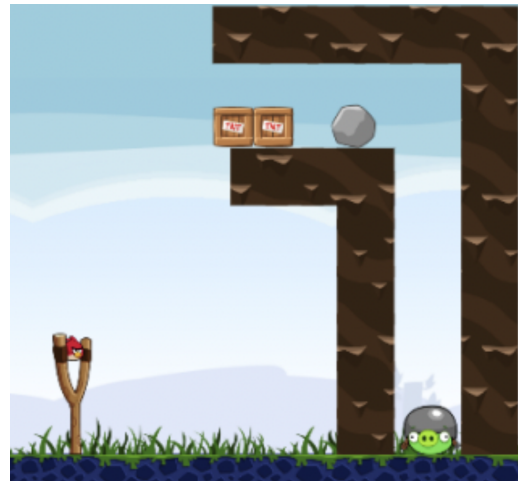
This year's task inspiration: **Crazy Machines (2005)**.

PC-Game from 2005.
Features physical reasoning puzzles in the form of Rube Goldberg chain reactions.
Spiritual successor to the classic DOS game,
"The Incredible Machine".

Background of the Task:

- Understanding and reasoning about naive physics is a fundamental skill for intelligent agents
- But solving physics puzzles remains a challenge for AI



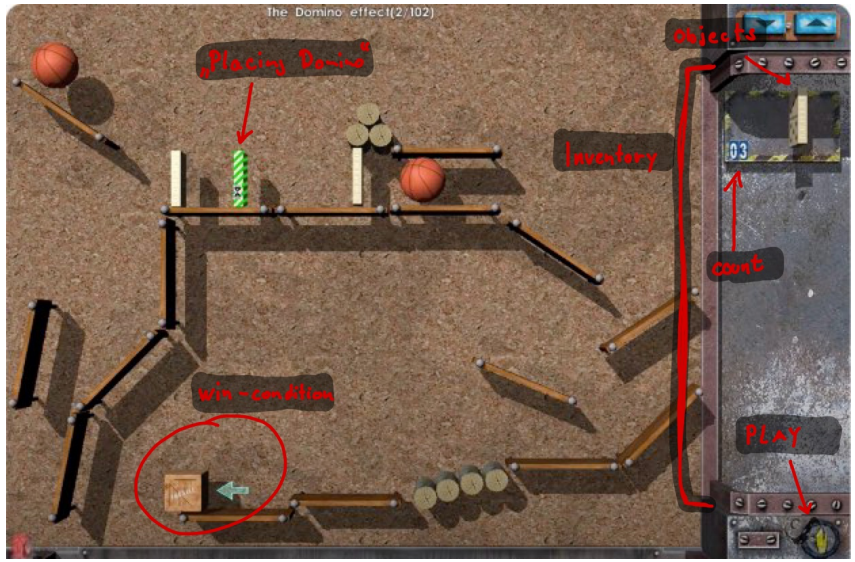


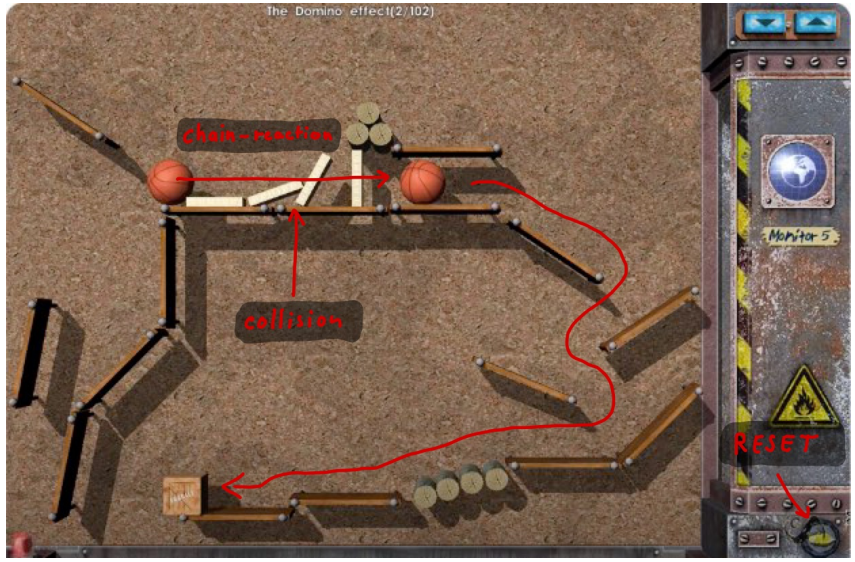
other physical reasoning benchmarks e.g. PHYRE.AI and AIBIRDS

Your Task: Develop a low-fidelity clone of *Crazy Machines* that may be used to test the physical reasoning abilities of AI agents.

In a Nutshell

- Each level starts with a partial contraption and a win-condition
- Player completes contraption by placing objects on screen
- if a chain reaction can satisfy the win-condition the level is won
- The heart and soul of the game is the **level editor** and the **underlying physics engine**[a].





[a] watch a walkthrough on youtube

Feature Overview: Core Features

Base of the Game. These features *have to be* implemented.

- Switching **simulation-** and **editor view** with play/reset button.
- Feature static, dynamic, and special **physics objects**[1]
- In **editor view** can placing/removing certain objects
- **Level loading** using human-readable JSON files
- **Puzzle mode** (as seen in Crazy Machines) with level progression
- **Simple graphic**s using basic shapes and colors, **real-time display**
- Levels feature **win conditions**, **constraint zones**, and **inventory**

Attention! Implementing (extra) features not listed or not serving the features listed below counts as a mistake in requirements engineering may be penalized.

[1] specific selection of objects see following slides

Feature Overview: Elective Features

Extensions to the base game. Choose *three* features to implement.

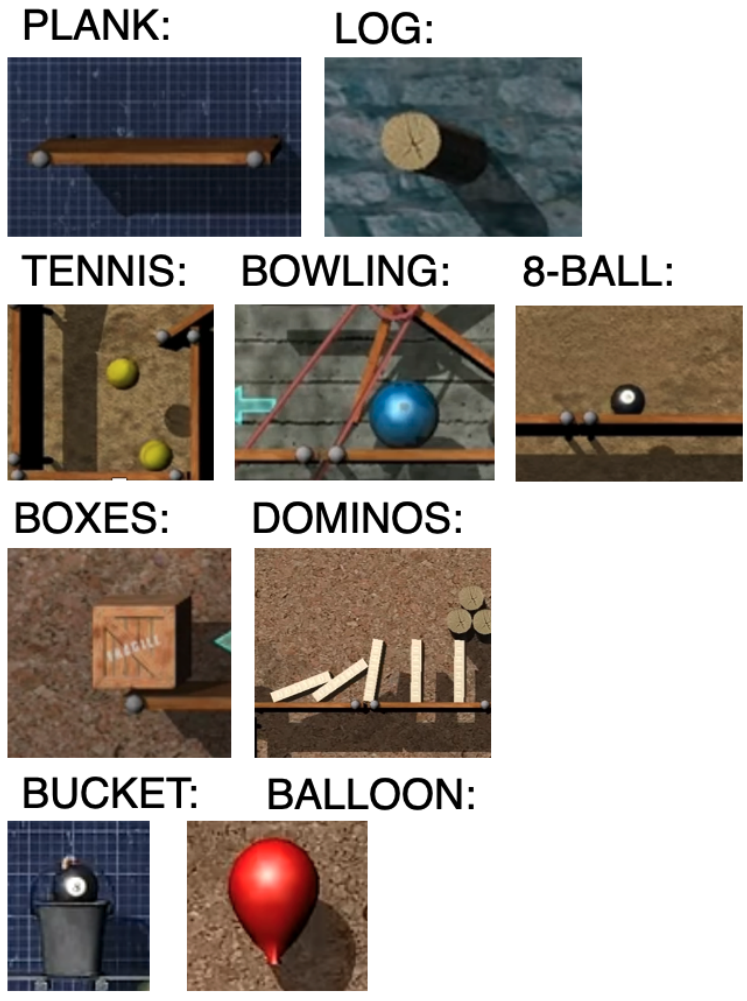
1. **Menus:** Add a main menu and a level selection screen with thumbnails and level info, mid simulation pause screen
2. **More Objects:** Extend the object set with ropes, anchors, levers
3. **Machine Interface:** Support a headless mode (no graphics), command-line interface, and game state logging for AI testing.
4. **Level Editor+:** Enable drag & drop, object rotation/movement, undo/redo
5. **Improved Graphics:** Render objects with textures/images, add background visuals, collision sounds

Feature Overview: Change Request!

After Release II there will be a *Change Request* for additional features.

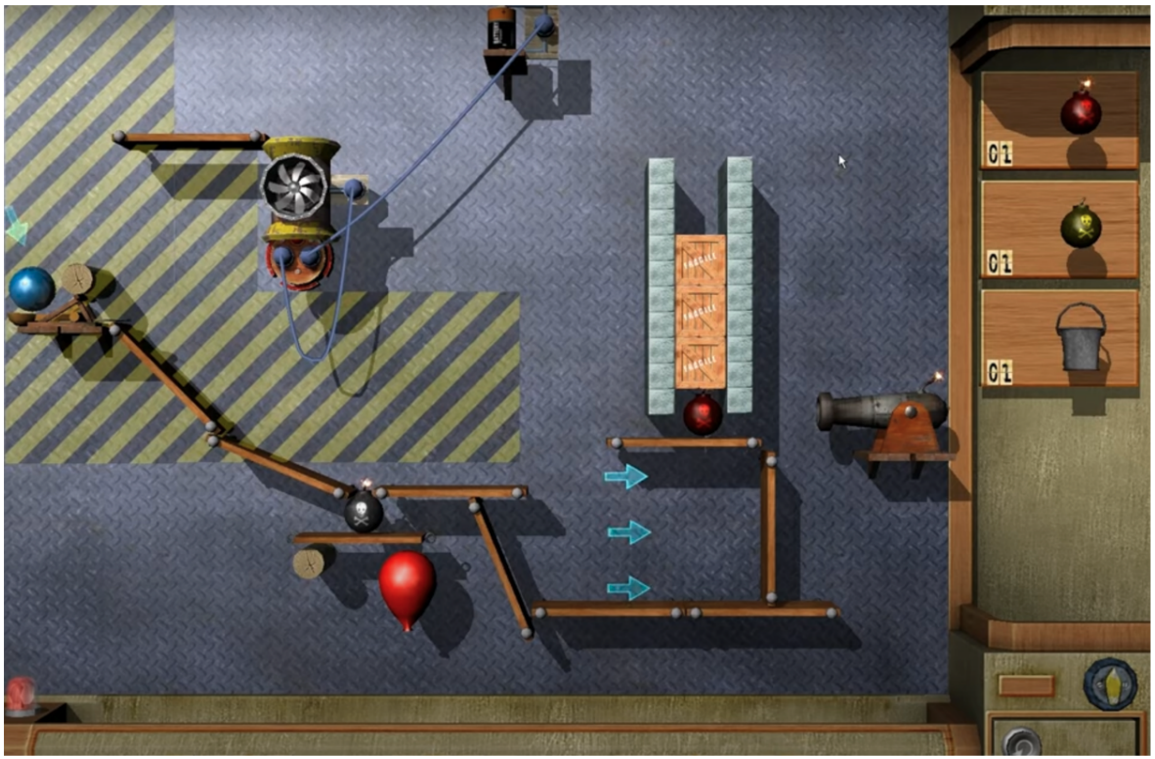
Subset of Objects from Crazy Machines

- **Static Objects:**
  Planks (long, thin, immovable, orientation matters), Logs (round, immovable)
- **Dynamic Objects:**
  Balls (default, bouncy, heavy), Polygons – Boxes (square), Dominos (rectangular)
- **Special Objects:**
  Balloon (floats up), Bucket (can hold a ball)



Puzzle Mode

RESTRICTION ZONE



INVENTORY

PLAY-BUTTON

WIN-CONDITION

Screenshot from the original game

Game Mechanics: Puzzle Mode

- **Inventory:** a set of *level specific objects* are available, in a limited number
- **Restriction Zones:** a object can not be placed within the zone
- **Win Condition:** display the win condition and check if it satisfies during simulation
- **Play/Reset:** Switching *simulation-* and *editor view* with play/reset button.

Standing on the Shoulders of Giants

Modern software development relies on powerful libraries!

- Use **JavaFX** for rendering and UI.
- For collisions and physics, a physics engine is essential. We provide **jBox2D**[2].

These libraries form the foundation for your game!

[2] Feel free to explore alternatives but support may be limited

Non-Functional Requirements

Code Quality: Adhere to defined coding style, metrics, and test coverage guidelines; use comments where necessary.
Test Coverage: 80% coverage (excluding GUI classes) via JUnit tests.
Documentation:

- Up-to-date requirements via story cards and use-case diagrams
- Architecture documented with UML (class-, sequence-, etc-. diagram) where appropriate
- User manual and Regularly maintained Javadoc

Detailed description and recommendations on the course site!

Release I — Deadline: 21.05.

Goal: Initial planning, requirements, and task assignments completed. Team gains experience via sanity checks.

Organization: Team is registered in QIS. GitLab project is forked from the template and set up. All members are added to the project and have successfully cloned the repository.

Sanity Checks: Project builds and runs via Maven on all team members' machines. Team members have explored unfamiliar libraries or common tasks, and shared findings internally. Low-fidelity/code prototypes are expected (but not graded).

Blast-Off Documentation: Requirements are documented (e.g., via story cards). A project plan is created with timeline, milestones, and team responsibilities.

Release II — Deadline: 25.06.

Goal: Imagine presenting your progress to an investor to secure further funding. You must demonstrate a running build and selected core functionality.

Minimum Viable Product (MVP): The game is recognizable and a basic MVP demo is functional. A GUI exists, and core game and most of the chosen features are *there*.

Important Notes: This milestone is not graded directly, you will receive feedback. However, if the requirements are not met, you will be asked to fix issues within 10 days. Failure to do so may lead to removal from the project.

Release III — Deadline: 23.07.

Goal: The game is tested and fully playable. Game includs all core features, a sufficient selection of elective features, and the additional features.

Final Submission: The entire project must be submitted via GitLab, including all source code, documentation, artifacts, etc.
Required PDF documents in the project root: `Manual.pdf`, `Requirements.pdf`, `Architecture.pdf`

Final Presentation: A 20-minute presentation showcasing your product, highlighting core functionality and any unique aspects (w.r.t. software engineering).

Reminder: Each release, including the final one, must be tagged in the Git repository (`r1`, `r2`, `r3`) to mark for evaluation.

Support

Weekly Tutor Slot: Each group is assigned a tutorial slot every Wednesday (**12:00–15:00** or **15:00–18:00**, depending on the group) in PC Pools 1–4, Building 64. You are expected to meet *your tutor* weekly, report progress, and discuss issues.

Moodle-Overflow: Peer-support forums for cross-group collaboration. Helpful activity (answers or documented solutions) may earn **bonus points** awarded at the end of the course.

Student Union (Fachschaft): The student union is available for support, feedback, or just to exchange experiences. They will also run a survey to help evaluate the course!

Tutors Contatct Information

- **Annabell-Chira Blaumann:**
  annabellchira.blaumann@student.uni-luebeck.de
- **Jennifer Grochola:**
  j.grochola@student.uni-luebeck.de
- **Roman Schierholt:**
  roman.schierholt@student.uni-luebeck.de
- **Alex Storjohann:**
  alex.storjohann@student.uni-luebeck.de
- **Mahdi Wahizzada:**
  mahdi.wahizzada@student.uni-luebeck.de
- **Lennart Schmidt:**
  le.schmidt@student.uni-luebeck.de

How to get Started?

- Find a Team!
- Go to the On-boarding Section in the course Site! Get familiar with the Template and Libraries and DevOps-Tools.
- Analyze the Requirements, what implications will they have on your software? Test out Tools for creating documentation like Draw.io.
- Research aka. *Google like crazy*. Learn about the architecture of other software.
- Set up your IDE, Git, Workflow & Testing. How to do specific things with the libraries. Start doing *Sanity Checks*.
- Find ways to organize as a Team, Trello-Boards, Tickets, Miro, Overleaf, Responsibilities, etc.

Common Mistakes to Avoid

- Skipping Git commits or working without branches.
- Delaying implementation until the last minute (risking potential GitLab downtime).
- Writing little or no documentation.
- Being afraid to ask for help.
- Ignoring feedback from tutors or teammates.