

TECHNICAL UNIVERSITY OF DENMARK



---

## (02160) Agile Object-Oriented Software Development

---

LASER MAZE

Group 3

Adam Hossein Winther  
(s240844) - s240844@dtu.dk

Amin Aïssa Amajjan  
(s224529) - s224529@dtu.dk

Hugo Demule (s231675) -  
s231675@dtu.dk

Léonard Amsler (s231715) -  
s231715@dtu.dk

Lina Mounan (s231865) -  
s231865@dtu.dk

Nathan Gromb (s231674) -  
s231674@dtu.dk

April 30, 2024

## Demo video

<https://drive.google.com/file/d/1jfmIlx74CWsay8k9CaRvS5pqWYZPRB2d/view?usp=sharing>

## Repository link

<https://gitlab.gbar.dtu.dk/02160-f24/group-3>

PS: If you would like to play the game starting from our repository, please follow the README.md file in the gitlab repository.

## 1 Introduction and project description

This report has been made as part of the course *Agile Object-Oriented Software Development* and focuses on our implementation of the game Laser Maze. We will present an analysis of the methodologies and theories we applied and how they relate to agile and object-oriented software development principles. The purpose is to clarify how and why these specific approaches were chosen to support the development of the game.

This section outlines the optional and mandatory functionalities of the project, detailing the interpretation of each requirement:

### Mandatory functionalities

- M1: Game and rules  
"The system should capture the essence of the game into an object-oriented system, thus giving the opportunity to interact with the system and play the game according to the expected rules (i.e., the game should not allow for wrong behavior)"
- M2: User interface and interaction  
"It should be possible to interact with the game via a graphical user interface. The graphics should not be advanced."
- M3: Game modes  
"The game should support different game modes. [...] A random level (given a certain difficulty) mode as well as a "campaign" mode should be supported (in campaign mode, the player should be able to continue until they fail one level)."

### Optional functionalities

- Persistency layer
- Improved graphical representation
- User authentication
- Sandbox for creation of level
- Sound system

The primary focus was initially placed on the mandatory functionalities, including *Game and Rules*, *User Interface and Interaction*, and *Game Modes* as outlined previously. From this groundwork, further development was made to incorporate optional functionalities. Additional features suggested in the project description, included the *Persistence Layer*, *Improved Graphical Representation* and *User Authentication*. Consequently, the decision was made to add personal functionalities such as a *Sandbox for Creation of Levels*, a *Sound System* and more. These enhancements were chosen to enrich the gaming experience and create a more interactive and personalized environment.

### 1.1 Logic Behind the Selection of Functionalities

To capture the essence of the game using the object-oriented paradigm, objects are designed to represent the game's components such as the tokens and laser. The main goal of the game is to

place tokens in the level and redirect the laser to reach the target to move on to the next level. The system ensures enforcement of the game rules, thereby preventing any illegal behavior, like moving a non movable token on the level. To check if the level has been completed, it has to make sure that all the tokens are placed, the checkpoints passed (if there are some) and the target hit by the laser at the right orientation.

Swing is used for the implementation *Graphical User Interface* (GUI) of the game. The GUI enables players to interact with the game: navigating between the menus, moving the pieces in the level, generating and visualising the laser. To improve on the basic GUI we added some animations and transitions at the end of a level.

Additionally, three game modes are put in place. *Campaign Mode* where the player has to progress as far as possible through increasingly complex levels. A *Sandbox Mode*, where the player can create their own level, edit it and then play. The third mode is the *Random Mode*, which plays a random game in the campaign. You can also choose a random video from a specific set of Levels.

On top of that, the *User authentication* is also implemented in the game, enabling players to create individual logins backed up with a hashed password. Along with the persistence layer, it allows us to retrieve the progression of the campaign of the user.

A *persistence layer* was also integrated. It allows to store the levels created in sandbox mode and as previously said the user's progression through the campaign. The components of the levels are stored in JSON formats for efficient state management and campaign progress and campaign progression in a .txt file. The choice of file-based persistence over databases allowed more simplicity and ease for the deployment. This approach ensured that the game state could be saved and subsequently restored, allowing for a seamless continuation of play and enhancing user experience by maintaining continuity.

Finally, a comprehensive *Sound System* was developed to enhance the user interface and game-play experience. This system included background music for the menu, as well as sound effects for tokens (game-piece) movements and menu interactions such as reset and back actions. This audio framework was structured across three classes: a central *Sound System* class to manage audio playback, a *Sound Path* class containing references to the .WAV file names, and a *Sounds* class that compiled all the audio files. This modular approach ensured a robust and easily maintainable audio subsystem within the game architecture.

## 2 User stories

In this section, the focus will be placed on '*User Stories*'. We have separated our user stories based on each mandatory functionality and optional features.

### 2.1 User stories on the game rules (M1)

Those user stories are about the game rules. This involve having everything needed to model the game it its whole.

- As a player I want to have a level that contains tokens on a grid to have a playable game.
- As a player I want to move a token to play the game.
- As a player I want to be able to progress in a campaign mode to make progress in the game.
- As a player, I want to be able to check if my solution is correct in order to end the game.

### 2.2 User stories on user interface and interactions (M2)

Those user stories are about the need of a graphical interface for the user to play the game.

- As a player I want to be able to have a graphical interface to navigate through the different game modes.

- As a player I would like to be able to play a level thanks to a graphical interface.

## 2.3 User stories on game modes (M3)

These user stories tackle the different game modes that should be implemented.

- As a player I want to be able to start a random game to test my ability at any difficulty.
- As a player I want to be able to progress through a campaign of increasing difficulty.

## 2.4 User stories on optional features

Our optional features can be highlighted by this list of user stories.

- As a player I want to be able to create levels to open a world of unlimited possibilities.
- As a player I want to be able to register my account to keep track of my progression.
- As a player, I want to have a soundtrack while playing to the game.
- As a player, I would like to have a better graphical interface such as animations to enjoy the game even more.

# 3 Software Architecture and Design

## 3.1 Classes organization and architecture

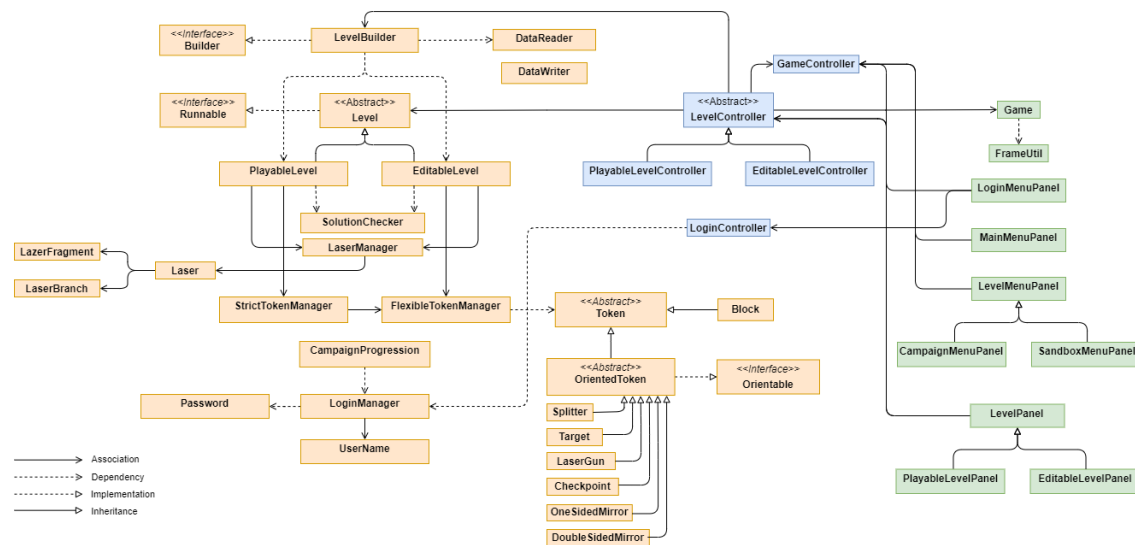


Figure 1: Simplified UML of the Project. **Orange**=Model, **Blue**=Controller, **Green**=Vue

## 3.2 Design Patterns

When faced with software design challenges it was practical to fall back on design patterns to tackle the problem. Here's some of the design patterns we implemented:

- **Model-View-Controller** : The general architecture is based on this pattern. The model is in charge of the logic of the game. The View part is in charge of the whole GUI. Then the controller is the bridge between the two: It handles the View's input and updates the Model accordingly. The model never interacts with the view directly, only through the controller.
- **Adapter**: `DataWriter` and `DataReader` help with creating JSON files from a `Level` and the other way around.

- Builder: `LevelBuilder` allows to create `aLevel` easily without bothering with too many parameters.
- Proxy: `StrictTokenManager` has a `FlexibleTokenManager` and controls the access to it. We used this pattern to avoid code duplication as well as adding features on the original `FlexibleTokenManager`.

### 3.3 S.O.L.I.D Principles

#### 3.3.1 Single Responsibility principle

Ensuring as much as possible that each class has one clear and defined responsibility. For example, we decided to implement a `TokenManager` to handle the transfer and movement of the tokens rather than leaving that control to the level. If it did happen that a class was taking too much responsibility, some refactoring was made. That's why we have a `LevelBuilder` class to create a level.

#### 3.3.2 Open/Closed Principle

Basing the addition of new functionalities on abstract classes or interfaces. For instance, the optional implementation for the sandbox mode, this new type of level is a subclass of the abstract class `Level`. Allowing for additional new features without modifying the existing code.

#### 3.3.3 Liskov Substitution principle

Instances of a super class should be replaceable with instances of a subclass without affecting existing behavior. To respect this principle we mostly ensured that inheritance made sense, and if not to use more abstracted class. In particular, we separated the orientation characteristic from the basic token and created an `OrientedToken` that implements the `Orientable` interface instead.

#### 3.3.4 Interface Segregation Principle

Making sure that the created interfaces do not burden the classes that implement them with unused properties by mostly keeping them on the smaller side.

#### 3.3.5 Dependency Inversion Principle

As explained earlier, this is again linked to abstractions (*high-level modules should not depend on low-level modules*) and we kept in mind to first create abstract classes or interfaces before specific implementations.

## 4 Testing

### 4.1 Description of our testing protocol

Here is a description of our testing process for our model. At first, we brainstorm as a group to get the main ideas for the feature we want to develop. We then split the feature into multiple needed scenarios. And only after we specify one person (or group of person) to develop the actual implementation to make the tests pass.

#### 4.1.1 Example

This is an example, part taken from our project (`TokenPlacement.feature`):

Scenario: Successful

Given I have a level that contains an empty board

When I try to place movable token at position (2, 2)

Then A token should be placed at position (2, 2)

And The board should reflect the change

In this case, if the automated scenario fail initially, we then refactor it to proceed to implement the scenario. The scenario is considered complete once the automated scenario passes, indicating that you're now allowed to pass over and create a new scenario. A user story is completed when all examples in the scenario are automated and passed.

## 4.2 Testing coverage and success rate

The cucumber tests entirely cover classes of the model part (with 85% of line coverage). The success rate is (100%, as we can see by starting all the cucumber features.

Here is the proof of the 100% success rate:

```
156 Scenarios (156 passed)
689 Steps (689 passed)
0m0,525s
```

## 4.3 UI testing

In order to ensure the correct functioning of our UI, we launched the game after making some changes and performed actions that could highlight potential issues (for instance, clicking on a button that we just modified, or completing a level to verify the correctness of the campaign progression).

# 5 Project management details

In this section, we will outline our work progress, detailing our approach to sprints the use of Kanban board via Trello, pair programming, commit practices, and testing strategies.

## 5.1 Description of the project management

We decided to have sprints of 1 week where at each meeting we defined and prioritized the task to do on our Trello.

## 5.2 Pair Programming

At the beginning of the project, we initially employed pair programming while working with the user stories and applying BDD-TDD testing methodologies particularly during sprints (meetings). After three days of collaboration, we began to work more independently due to the division of task and focusing on different classes. However, pair programming remained an essential strategy during sprints in pairs. It proved particularly valuable when team members encountered issues with their code or specific classes. Given that we were a group of six, the use of pair programming was intermittent, adapting to the needs of the project and team dynamics. This flexible approach ensured effective problem-solving and knowledge sharing within the team.

## 5.3 Version control strategies

The project was managed with the Gitlab tool using the git technology. We structure our branches structure as such:

- Main branch: This is the branch used for really important event such as the final version of our game. Merges to this branch should only come from dev.
- Dev branch: This is the most up-to-date functioning branch. We tried to maintain a stable version of our game into this branch by limiting the bugs. Merges should come from "feature" branches.

- Feature branch: When creating a new feature or changing something on the dev branch, new feature branches are created. Ideally those branch are maintained by only one person to avoid conflicts.

We have decided to restrict any push to dev and main from an IDE (by modifying the branch options in gitlab) to avoid mistakes and ensure that any merge to those branches are made through a merge request and that validation and code review is made.

A git manual has been made at the beginning of the project to ensure that everybody had the basic knowledge to contribute to the project via git.

## 6 Work distribution

	Léonard	Lina	Amin	Adam	Hugo	Nathan
Workload	1	1	1	1	1	1

Table 1: Workload Distribution

This number is based on the best that each member could have done, taking our respective experience before joining on the project into account.

## 7 User Manual

Welcome to the user manual! Here you will find out every details you need to know to play the game in its best and appropriate way.

This user manual is divided into multiple section, each of them about one window-type you can face while playing the game.

### 7.1 Login menu

The first window that you will face when running the game is the login menu. Here you can either register a new account or login to an existing account.

In order to register, press on the white placeholders to write your username and your password twice. Don't worry your password won't be read and stored as such! Then press on the "register" button.

To login, also press on the white placeholders to enter your credentials. Then press on the "login" button.

You may encounter some error message if :

- You try to login with a username that has never been register
- You try to login with the wrong password for a username
- You try to register with a username that has already been registered
- You try to register with an empty username or password
- You provide two different passwords while registering

### 7.2 Main menu

After having connection your account, you will be directed to the main menu. This main menu contain several elements:

- A "logout" button to logout your account. You will be redirected to the login menu.
- A "Campaign" button in order to go to the campaign menu

- A "Sandbox" button which makes you redirecting to the sandbox menu for creating your own levels!
- A "Random" button to play a random game among the campaign levels.

### 7.3 Campaign menu

After having clicked on the "Campaign" button from the main menu you will be redirected here.

You will see all campaign levels to be played in the campaign. A black level would mean you have to progress through the campaign to have access to that level. A green one is a level that is available to you (it is either an already finished level or your current campaign progression).

By clicking on one of these square, you will be redirected to the level itself.

You can decide to come back to the main menu back clicking on the arrow on the bottom of the screen.

### 7.4 Sandbox menu

After having clicked on the "Sandbox" button from the main menu you will be redirected here.

It contains all these elements:

- A red bar with a "+" to create your own level from scratch!
- (If you have already made some levels) Under the red bar, you will see all the levels already created.

Each level has:

- Its date of creation to distinguish the levels
- A button to play the level, by clicking on it you will be redirected to the level itself.
- A pencil to modify this level again, by clicking on it you will be redirected to level modification window.
- A bin to delete the customized level.

You can scroll among all level if more are provided that the size of the window.

- A back arrow at the bottom of the screen to go back to the main menu.

### 7.5 Level window

To resolve a level, you will have to drag the tokens (with the mouse) from the inventory (on the bottom section) into the board at the wanted position. Don't be scared you can move them again later.

There is a key difference between the tokens that are already placed at the beginning and the tokens that need to be placed. You won't be able to move and rotate the ones that were already placed. They are part of the level for the eternity! A distinguishable visual has been made to help you know which tokens you can move, the background tile of those tokens contain a hashed border.

Rotation of tokens is really crucial for resolving a level and can be achieved by clicking on the token. As previously said you can only rotate the tokens that are in the inventory from the beginning.








In order to remove a token from the board, you can drag it into the chest at the right side of the screen to place it back to the inventory.

On the top right corner, you will find two buttons. The one on the left redirects you to the campaign menu and the right one reset the current level by placing all movable tokens into the inventory.



### 7.5.1 Description of the tokens

Here is a list of all the tokens from the game along with their graphical icon.

-  Laser gun - This is where the laser starts from. It is oriented in the direction of the red line.
-  One sided mirror - It redirects the laser at 90 degrees and is only reflective on the red side.
-  Double sided mirror - Same as the one sided mirror but it is reflective on both sides.
-  Receiver - The target of the laser, it needs to be hit to solve the level
-  Checkpoint - This is token where the laser should go through to validate the solution. It is orientable and needs to be hit on the appropriate direction
-  Blocker - No laser go through this token
-  Splitter - This is a complex token where the laser is fragmented into two part: the laser goes through and is reflected at 90 degrees as the mirrors

## 7.6 Level editor (sandbox)

In the sandbox you will get a 7 x 7 empty board in the center of your screen. On the left side of the screen, you get all tokens that you can decide to add to your level. By clicking on them, they will be added to the inventory. You can add as many tokens as you want the only restriction is that only one laser gun could be created.

After having placed the tokens in the inventory, you can drag them to the wanted position or let them into the inventory for the player to actually place it.

You can also drag a token (from the inventory or the board) to the bid on the bottom left corner of the screen to remove a token from the level.

On the top right corner, you will find two buttons. The one on the left redirects you to the sandbox menu and the right one place all placed tokens into the inventory.

You can refer to the section 'Description of the tokens' to get information about the different tokens.