trAIner
Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2. Architectural goals and philosophy

The following architectural goals need to be considered during design:

- user friendly UI
- eye pleasing design
- different screens to play and to create the maps
- smooth visualization for AI / player movements
- offline functionality
- local storage of maps
- store leader board online
- read local maps
- run AI algorithm in background → multi threaded
- configuration of AI hyperparameters in sidebar
- automated unit test
- separate configuration panels
- futuristic look
- play levels with keyboard
- represent map by matrix of different blocks
- different behavior for different blocks on the map

3. Assumptions and dependencies

- From users perspective playing the game should be intuitive.
- In the same time the game must be interactive. The more interactive the game is, the less intuitive it becomes.
- Java runs cross platform so the game will be implemented in Java
- Due to team's low experience in development of projects and time constraints.
- Most of information about AI is in English, therefore we assume that most of users know English, therefore the default language is English
- The budget of the project is 0 euro, and theoretically could increase to 50. Therefore we will use a free hosting server

4. Architecturally significant requirements

- user friendly UI
- different screens to play and to create the maps
- multi - threading support
- store leader board online
- playability / fun factor

5. Decisions, constraints, and justifications

- The maps will be saved in csv format
- We add an option for a user to try playing the map himself as it gives makes the game more fun and

interactive. In addition creates a better understanding of an AI task

- The game should be able to be played offline. Therefore we will use local storage of the maps

6. Architectural Mechanisms

Artificial Intelligence Mechanism

The user will train his players with his custom-built maps based upon the AI algorithm. Furthermore the user will be able to adjust some hyper-parameters to improve his AI.

User interface

Displaying user friendly interface with smooth animations & no flickering.

Map building mechanism

User can build custom maps which will be used for the training of the AI. The maps will then be saved & can be reloaded at any given time by the user.

7. Key abstractions

- Map Builder Canvas – the logic behind how user can build a custom map
- Game Loop – make the game always running
- AI – the algorithm used to train the AI
- Map storage – how to store & load custom-built maps by user on local device
- Interface – interactive & easily understandable GUI

8. Layers or architectural framework

Game Loop

Continuously **runs the game**, is responsible for **processing the user input**, u**pdates the states** of the game, tracks the time and **controls the framerate**

Double Buffer

A **buffered class** encapsulates a **buffer**: a piece of state that can be modified. This buffer is edited incrementally, but we want all outside code to see the edit as a single atomic change. To do this, the class keeps *two* instances of the buffer: a **next buffer** and a **current buffer**.

When information is read *from* a buffer, it is always from the *current* buffer. When information is written *to* a buffer, it occurs on the *next* buffer. When the changes are complete, a **swap** operation swaps the next and current buffers instantly so that the new buffer is now publicly visible. The old current buffer is now available to be reused as the new next buffer.

Update Method

The **game world** maintains a **collection of objects**. Each object implements an **update method** that **simulates one frame** of the object's behavior. Each frame, the game updates every object in the collection.

Type Object

Define a **type object** class and a **typed object** class. Each type object instance represents a different logical type. Each typed object stores a **reference to the type object that describes its type**.

See DomainModel_with_layers.png

9. Architectural views

See ClassDiagram.png