

# Technisch Uitwerkingsplan



Door: Bastien Olivier Dijkstra, Daniel Bergshoeff

# Inhoud

|                                 |          |
|---------------------------------|----------|
| <b>Inhoud</b>                   | <b>1</b> |
| <b>Code structure</b>           | <b>1</b> |
| UML Diagram                     | 1        |
| Toelichting                     | 1        |
| Plan van aanpak                 | 2        |
| <b>Re-design code structure</b> | <b>2</b> |
| UML Diagram                     | 2        |
| Toelichting                     | 2        |
| Singleton Pattern               | 2        |
| Object Pooling                  | 2        |
| Decorator                       | 2        |
| Plan van aanpak                 | 2        |
| <b>Attachment</b>               | <b>2</b> |

# Code structure

## UML Diagram

[Class Diagram](#)

[UI Activity Diagram](#)

[Game Activity Diagram](#)

## Introduction

Our plan is to make a multiplayer tetris game, in which the way you play on your own field influences the other person's experience. Initially, the structure we had set up for creating our game had a few aspects that could be improved by making use of design patterns, which we will go over in this document.

Currently we have the following aspects:

- Block  
A block is the original shape that all tetris tiles are made of.
- Tile  
A tile is the complete shape of the 4 blocks melded together.
- Tile Manager  
The tile manager is used to keep track of the entire field and the blocks contained within it and is separate for each user, just like the field is. It also checks whether the player tile can move down, left and right or can be rotated.
- Controller  
The controller defines the keybindings of each player, so we can edit these before the game starts.

## Plan of action

For creating our Retris game, we decided we'd like to make use of Unity. The reason we'd like to use Unity is because it is free and adds to a fast prototyping workflow.

In Unity, we'll be making use of the features MonoBehaviour and Time.

# Re-design code structure

## UML Diagram

[Class Diagram](#)

[UI Activity Diagram](#)

[Game Activity Diagram](#)

## Introduction

### Singleton Pattern

The first Design Pattern we'd like to incorporate in our game is the Singleton Pattern. We'd like to implement our Tile Generator as a Singleton, because the Tile Generator only has a single instance and only has to exist from the moment the start menu is over.

### Object Pooling

Another Design Pattern we could use in our game is the Object Pooling Pattern. Since our Tiles can be reused multiple times amongst the different users, we don't have to destroy every tile and then remake it, we can simply reuse them from a Tile pool that we save in the Tile Generator.

## Plan of action

For object pooling we will implement the following class, which will coincidentally also be making use of the singleton pattern.

- Tile Generator  
The tile generator creates tiles for the tile manager to add to the field once a tile has reached the bottom of the playing field. This way, both users get the same tiles and the game stays fair. This also means less Tiles have to be constructed, because each tile is reused by the amount of people in the multiplayer game.

## Priority

For our project, the order in priority is as followed.

### 1. Creation and keeping track of blocks

The number one priority for our game is that it is possible to create a tile (consisting of blocks) and that the tile is saved within a grid that we have created, which we can check to add commands such as moving left and right or rotating the tile.

## 2. Movement and controller for player tile

Once our grid is up and running we can add our player tile and generate controls, so we may move the tile left and right and rotate it as we please.

## 3. Clearing a row once it is filled

As the player can now control the tiles, it is crucial that once a row is filled with blocks, the blocks are deleted and the rest of the field is moved down a block.

## 4. Adding multiplayer functionality

Once tetris works as it was intended, we shall add multiplayer functionality, making it possible to add players and have multiple fields depending on the amount of players added.

## 5. Adding multiplayer interactions

When the entire game is functional with multiplayer functionality, we may add interactions between the players. For example, special blocks may spawn that influence the experience of other players, such as increase the movement speed of their tiles or block their rotations.

## 6. Adding a menu and end-game screen

The final addition is to add UI for starting the game and a nice overview of how you did at the end of the game.

## Attachments







