Emergent Architecture Design

Group 5

Team Members

Liam Clark	4303423
Mitchell Hoppenbrouwer	4243889
Martin Koole	4217500
Ike Rijsdijk	4294106
Jorai Rijsdijk	4282264

1. Introduction

This document will have a high level description of software and hardware components of our games project.

1.1. Design goals

- **Responsiveness**
  The input of the player should be processed within a reasonable time. If it takes too long, the players won't have the feeling of satisfaction when creating their art. It will cause frustration, so it is an important design goal. The game needs to feel responsive.
- **Easy use**
  Players should be able to play the game easily. It doesn't require any hardware on the player's end. It has no learning curve, the player should be able to play the game instinctively.
- **Audience**
  Our audience consist of people in public places with spare time on their hands. They can be of all ages. We want the design the game to have people take their time. We don't want to rush them.

2. Software architecture views

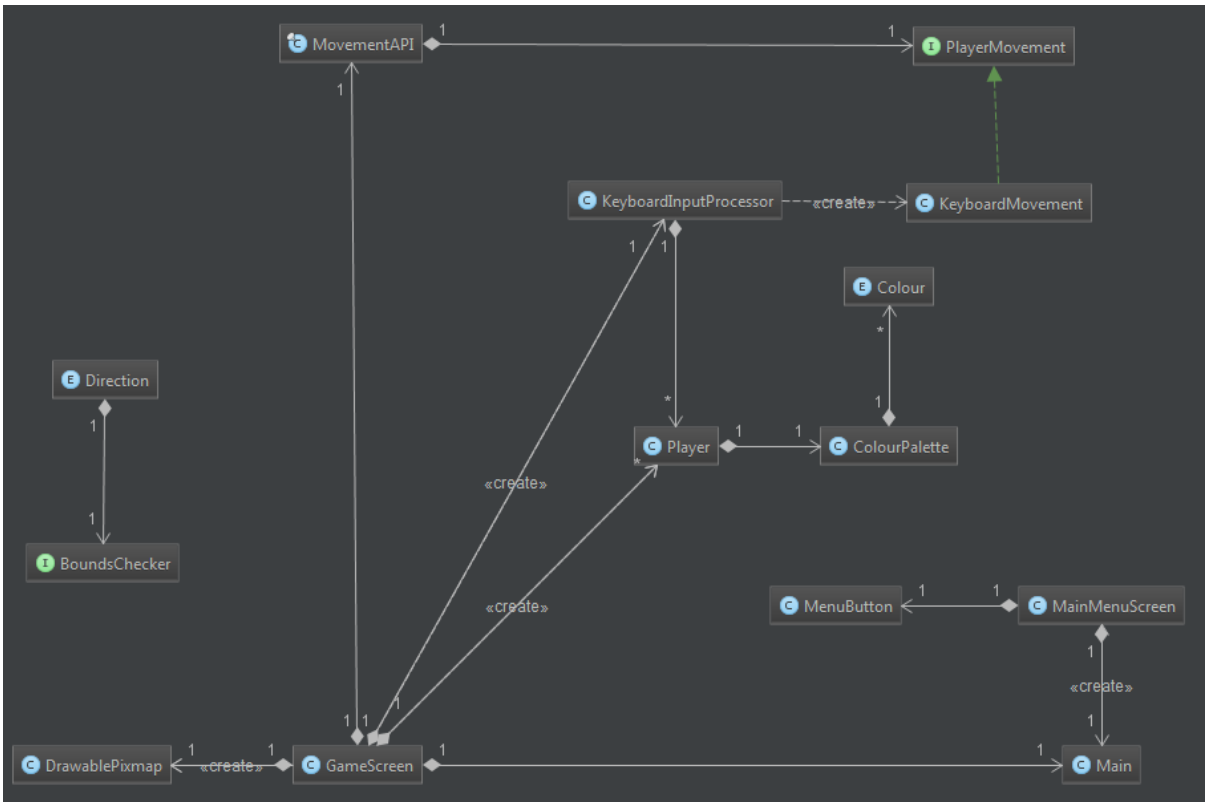
This chapter discusses the architecture of the system. The system is first decomposed into smaller subsystems and the dependencies between the subsystems are explained. In the second paragraph the relation between the hardware and software of the system is elaborated. The third paragraph illustrates the data management of the system.

2.1. Subsystem decomposition (sub-systems and dependencies between them)

The subsystems are simple. We want them to do a job and do it well. The core layer and the rendering layer are intertwined by LibGDX, but separate enough to see as different layers.

- **Image Processing Layer**
  All player input comes from processed camera input of the player this layer should parse the input to an understandable format for the core layer.
- **Keyboard Input Layer**
  Instead of using image processing, a keyboard can be used to paint. This will be used to test the game while the image processing is being developed. When image processing is ready it will replace this layer. This package contains the package *input*. This layer is inside the core layer due to the LibGDX structure. It will be removed once image processing can replace it.
- **Core Layer**
  The input from either the image processing layer or the keyboard input layer is converted to movement of the player. These player movements will be send to the rendering layer. This layer contains the packages *entities* and *config*.
- **Rendering Layer**
  The canvas of the painting needs to be rendered and changes need to be applied every cycle. This layer contains the packages *rendering* and *screens*.

We use a model-view-controller design pattern. In our design the rendering layer is our view. The image processing layer is the controller and the core layer is the model.



The above figure show our classes and their dependencies. The classes *MenuButton*, *MainMenuScreen, GameScreen* and *DrawablePixmap* are part of our rendering layer. The keyboard input layer consist of *MovementAPI, PlayerMovement, KeyboardMovement, Direction* and *KeyboardInputProcessor*. *Player, Colour* and *ColourPalette* belong to the

core layer. As mentioned before, the image processing layer will replace the keyboard input layer. For this reason the image processing layer is not shown in the figure.

What is really important to us is the simplicity of the structure. The game has only a couple of main objectives to run. Detect the player using a camera. Paint on the canvas accordingly. And display what is being created by the player. These are three things, image processing, core game functionality and rendering. Due to these main objectives we decided to divide each into it's own layer. We also want the possibility to run the image processing on a different system, because of it cost a lot. So it's important that it has it's own layer. The core game functionality is to receive the movement, assign a colour and send it to the rendering layer. In this layer we want the ability to also blend and change colours. The rendering layer takes all the output and displays it. This is a separate layer because when something is not showing up on screen we know where we can look.
We also have the Keyboard Input layer. This is separate because we want to remove it. By it being a separate layer we can disconnect it for the core layer without to much trouble. If it was in the core layer itself, the removal would be a larger task.

2.2 Hardware/software mapping

- **Camera**
  The camera observes the player in the playing field. It sends all its output to the image processing system. The camera will have to be installed beforehand and not by the player. This to ensure our design goal of easy use.
- **Image Processing System**
  The image processing system takes the output from the camera and processes it to detect players entering or leaving, detect brushstrokes being made by players and their requests to change the current colour. It sends the information over a LAN network to the core system. The image processing cost a lot, which is why it is on a different system. By having it on a different system the game will feel responsive, which achieves one of our design goals.
- **Core System**
  The core system consists of the core layer and the rendering layer. It takes the image processing output and performs the detected actions in the game. It then renders the image and sends it to the beamer. While the keyboard input layer is being used as a replacement for the image processing layer, it will be on the core system. Since the cost of the keyboard input is less compared to image processing, this is not a problem.
- **Beamer**
  The beamer receives the image from the core system and displays it. The beamer will have to be installed beforehand and not by the player. This to ensure our design goal of easy use.