Technical University of Eindhoven

DZC20 Design for Games & Play II - Team Beta

Serious Game Design - Final Report: Jumper

Authors:
Team Beta

Course responsible lecturer:
prof.dr. B.A.M. Schouten BA

Last modified: January 15, 2015

Revision: 0.5

Team Member	Role	E-mail address	Student ID
Jeroen van Hoof	Lead Art Design, Front-End	j.m.a.p.v.hoof@student.tue.nl	0778486
Kevin Cleijne	Art Assets, Evaluation	k.c.g.a.cleijne@student.tue.nl	0779007
Pieter Kokx	Lead Programmer	p.a.kokx@student.tue.nl	TODO:NUMBER
Joep Klein Teeselink	Back-End Programmer	j.klein.teeselink@student.tue.nl	0816396
Twan van Schijndel	Level and Story Design	t.v.schijndel@student.tue.nl	0857767

TU/e Technische Universiteit Eindhoven University of Technology

Table of Contents

Introduction

This report covers the process of the creation of a Serious Game designed for programming courses. The goal is to provide an insight into the design philosophies, choices made and hurdles overcome. We will first discuss the basic premise of the game. Afterwards we provide and overview of the design concepts, realization and finally conclusions with individual evaluations.

Team βετα

1 Goal

We intend to create a serious game designed to peak interest in the Computer Science bachelor on the TU/e. As such we focused on the programming discipline, a key part in this education. Our game is intended for people who are looking for an introduction to programming and are interested in following a Computer Science education.
Our objective then is to create a sandbox-y puzzle platformer that teaches you the basics of a programming language through its mechanics. The game itself must be fulfilling to play, providing a reward system as you go which consists of acquiring new abilities in the form of programming commands, unfolding the story and encountering new and interesting locations with more difficult puzzle elements. When the player stops playing he should have a basic understanding of a programming structure, building up the concepts such as recursion, for-loops and choice branches with IF and ELSE statements.

1.1 Target Audience

The demographic is aimed at high school students who wish to follow a computer science education. They need not have a background in programming, but should they do they will progress trough the game quicker and arrive at more challenging stages more quickly, increasing the level of programming required as levels progress.

1.2 Team Introduction

Our team consists of 5 members - all following a computer science degree. @TODO: who are you?
Jeroen van Hoof is a web science student
Pieter Kokx is a @TODO
Twan van Schijndel is a @TODO
Joep Klein Teeselink is a @TODO
and finally we have Kevin Cleijne, who is also a software science in his 3rd year.

1.3 Role Distinction and Allocation

We define the following roles necessary for our game:

Art Design - Jeroen van Hoof, Kevin Cleijne
the art design makes up the aestetics of the game. We uphold a single standard art style to keep the looks consistant. We wish to create our own assets to make the game unique.

Front-End Programming - Jeroen van Hoof
Placement of game objects, art assets and interface design.

Level Design - Twan van Schijndel
Our game is a puzzle platformer and as such level design is critical. The levels need to be created in a way that keeps the challenging yet not too complicated, pacing is key.

Back-End Programming - Pieter Kokx, Joep Klein Teeselink The game is made in Phaser, a HTML5 engine used to create web-based games. @TODO: add some info plx
Evaluation and Testing - Everyone a game has to be iterated upon and bugfixed during its creation. It is imperative to detect bugs early and fix them, upholding structure and design principles. Any irregularities, defects and bad ideas need to be caught and adapted in order to make a functional and enjoyable game.

1.4 Setup

For our development setup we used a GitHub Repository, allowing every member to work on the project at once. Conflicting files can be easily combined to a single file and every modification is clearly documented via the GitHub client. Additions and alterations are traceable and revertible with ease.
The game design concepts are documented in Google Drive - we documented images and ideas while brainstorming. This allows us to look back on our work and recall discussions and ideas, saving the result of brainstorm sessions for later.
The communication channels we used were primarily via WhatsApp and Facebook. These let us schedule appointments for when to come together and attend people to ideas and progression. We also utilized the GitHub issue section to flag problems and assign them to team members to fix.

2 Game Concept

We aim to create a puzzle platformer that teaches players a programming language as they play it. We keep the player entertained with interesting environments, story progression and nice visual/auditive effects. We want to allow the player to experiment with the coding language, providing them with a 'common' solution as well as other ways to go about the problem. When the player gets a good grasp of the coding language he could find special methods that enables him in finding a new solution altogether, rather than following the standard procedures of puzzle solving. We want the player to think about his programming abilities and exploit them in the game to figure out new and interesting ways of solving the situation at hand. as the player progresses he encounters new and interesting levels, acquires new programming skills while advancing the story.
We use a basic flat design art style that keeps things simple and clear, allowing for easy distinction between background visuals and foreground objects/platforms. For easy access to the game we opted to create it in a HTML game engine called Phaser, which allows us to deploy the game online, no installation necessary.

2.1 Story

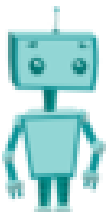

Figure 1: The protagonist: Jumper

Ultimately the game aims to teach the basics of programming - so what better protagonist than a robot? Jumper is a matrass-tester on the run from his overseers from the matrass factory. He breaks lose from his testing-chamber/prison and ventures out into the world. Luckily for him his jumping skills are quire adapt and he can easily traverse the treacherous areas outside of the facility. However that he soon finds that this is not enough and he has to rely on his new-found knowledge of the World-Programming installed upon him in his escape of the factory. Why is it that he can program certain objects? Where can he find more knowledge about this power? Who else is stuck in the factory? Who released him? Follow Jumper's quest and find out.

2.2 Gameplay Elements

The game allows for manipulation of the world around you using a programming terminal. Certain objects can be interacted with and re-programmed to move around, become solid or unsolid, activate or deactivate, ... the interactions increase as you progress, making things more interesting and difficult as you go. To not overwhelm the player we have introduced a help() command that provides you with insight into what is going on and what you can do, how it works and provide hints to a solution. You can think of the help command as a sort of catalogue explaining the workings of a particular programming function/method. The game requires some precision jumps and puzzle solving (via programming) which keeps things fresh. You can approach the problem in many different

ways. A player might be inclined to traverse the level, solving obstacles as he encounters them, or they may sit back and think about the level - program/solve it and then finally traversing the puzzle in one swoop. They can even think of ways to reach the end by using hidden functions at their disposal, which they can find by experimenting and knowing the programming language better. Picking up hints to these 'secret' methods throughout the game.

2.3 Level Design

Puzzle platformers require good level design to be fun to play. The levels need to be challenging and incorporate the different programming abilities via obstacles that should be logical elements and not feel very out of place. To achieve a good level we adhered to a standard of keeping objects as a power of 2, generally picking 32px by 32px for a standard platform, having the player character at 32x64. By adhering to this standard we can precisely place objects and platforms for the different levels. Special attention is given to the introduction/tutorial level(s),aiming to create a connection between the player and the playable character. The tutorial levels introduce new programming mechanics - the objective of what the player needs to do to progress is clearly visible, it is up to him to find out exactly how the command works - turning to the help function when in doubt, as well as tips displayed in the level. We wished to make the solution not very ¡do this¿ like but instead attempted to blend the solution in the background, having the player observe behaviour occurring in the game and learning from that.

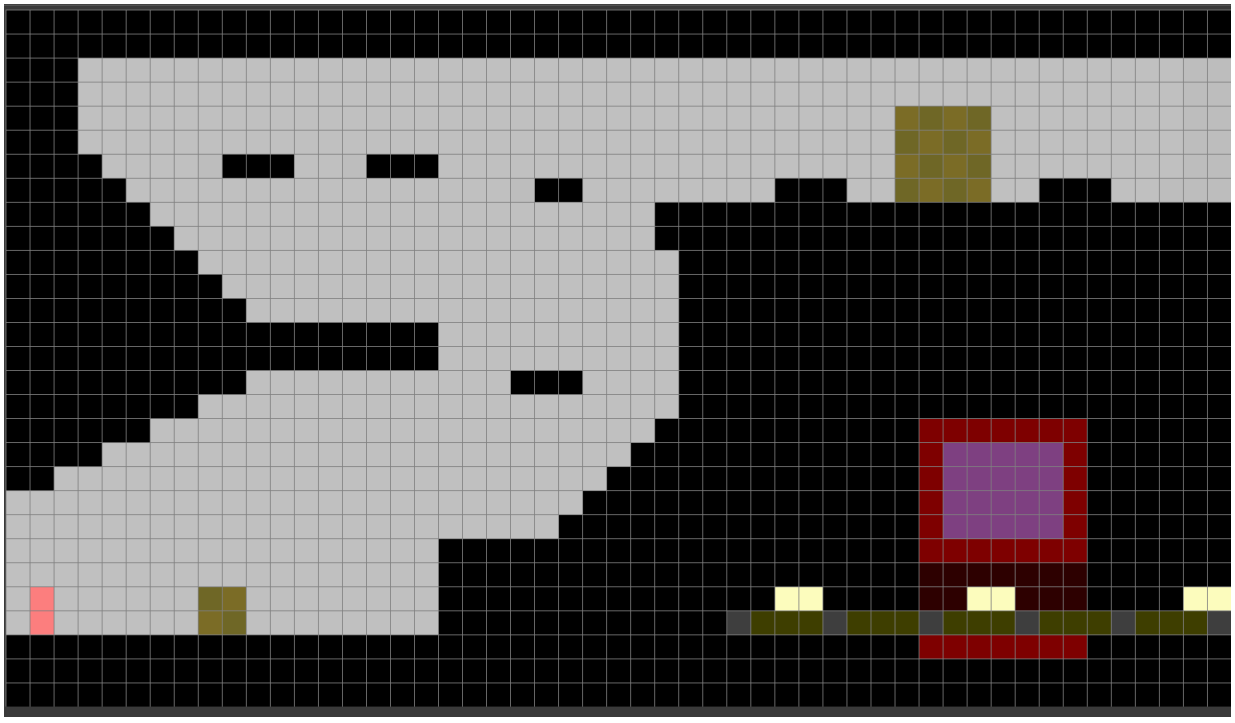

Figure 2: tutorial concept

3 Realization

Now that we have the game concepts in place, we need to actually create a game. We opt for a HTML game that can be deployed online for easy access. Platformers are not as resouce intensive which allows for this approach. Levels can be loaded fast and there is practically no downtime. This approach allows us more flexibiltiy, utilizing a HTML engine Phaser, as well as some CSS and Javascript to further enhance the game.

3.1 Structure Realization

3.2 Programming Habits

3.3 Game Interaction

The interactions the player has with the game are basic platforming with a jump combined with the puzzle aspect, solvable using the command terminal used to manipulate objects. The terminal allows for programming of game objects, calling upon readily made functions (and some hidden ones to allow for experimentation) that can change the world so that you may advance the level. Since we do not require players to have a programming background, we have to supply them with the know-how of these manipulations. To assist in this we need to have a tutorial for each of these new programming aspects. After such a tutorial we need to provide a codex to re-evaluate the just learned mechanic. We do this by means of a help() function. Upon calling the help while trying a manipulation we provide an insight into its workings and further explain the usage. To further assist the player in doing some tedious tasks we incorporated a click mechanic that will pre-write the object name whenever the player clicks on a manipulatable object in the game world, giving it a while border implying it is selected. This click auto-fills the terminal and makes it ready for manipulation.
To improve the gameplay and clarity of the game, we have plenty of visual and audible queues, giving a clear indication when something goes wrong or right, as well as an error report in the terminal - telling you what went wrong. We use screenshake for a more tactile experience while playing the game. Falling on objects has actual weight to it due to the shake, which increases the faster you crash into something. Each object has a landing sound attached to it - changing the effect depending on what type of surface you interact with.
Animated objects allow for a lively world that is interesting to look at. Some of them incorporate a mechanic, such as a bouncing matrass and conveyor belts, others are just visual effects designed to fluff up the world.

3.4 Level Creation

The levels have been designed on paper prior to implementation to the game. The conceptual drawings follow the 32px by 32px rule of a single block, allowing for a good scalable representation of the level and accurate platform distancing, to be traversed by Jumper. Conceptual levels introduced new mechanics which we then incorporated as programming functions for the game, such as having obstacles dissipate and become unsolid or having objects stand in the way of your level progression. These ideas were drawn and incorporated into a playable level, then created as functions and realized in the game for the player to solve.

3.5 Art Creation

The art assets were made by 2 members, and as such it was difficult to adhere to a single style, we used assets that were made in a way that it was not clearly obvious that two people created them. The style we used is called Flat Design. Even though it appears basic, there are subtle effects in play that require some effort to realized. As such we had to scrap a lot of assets, removing them entirely or having to heavily modify them to fit to the standard. The assets themselves are vector drawings, able to scale up and down without resolution loss. This allows us to change the scale of assets on the asset itself rather than use Phaser to manipulate scaling. As long as the assets adhered to the ratio rules we set, they would fit perfectly into the canvas. Even if this did not work out it was easy to change the scale on the asset itself, them being vectors instead of rasterized images.

4 Iteration

We started out creating the basic structure of the game, setting up a UML diagram which we adhered to. We had to do some renaming to make everything more logical when dealing with the code, having 2 different classes called Game is ambiguous and has been resolved by renaming one to Main.

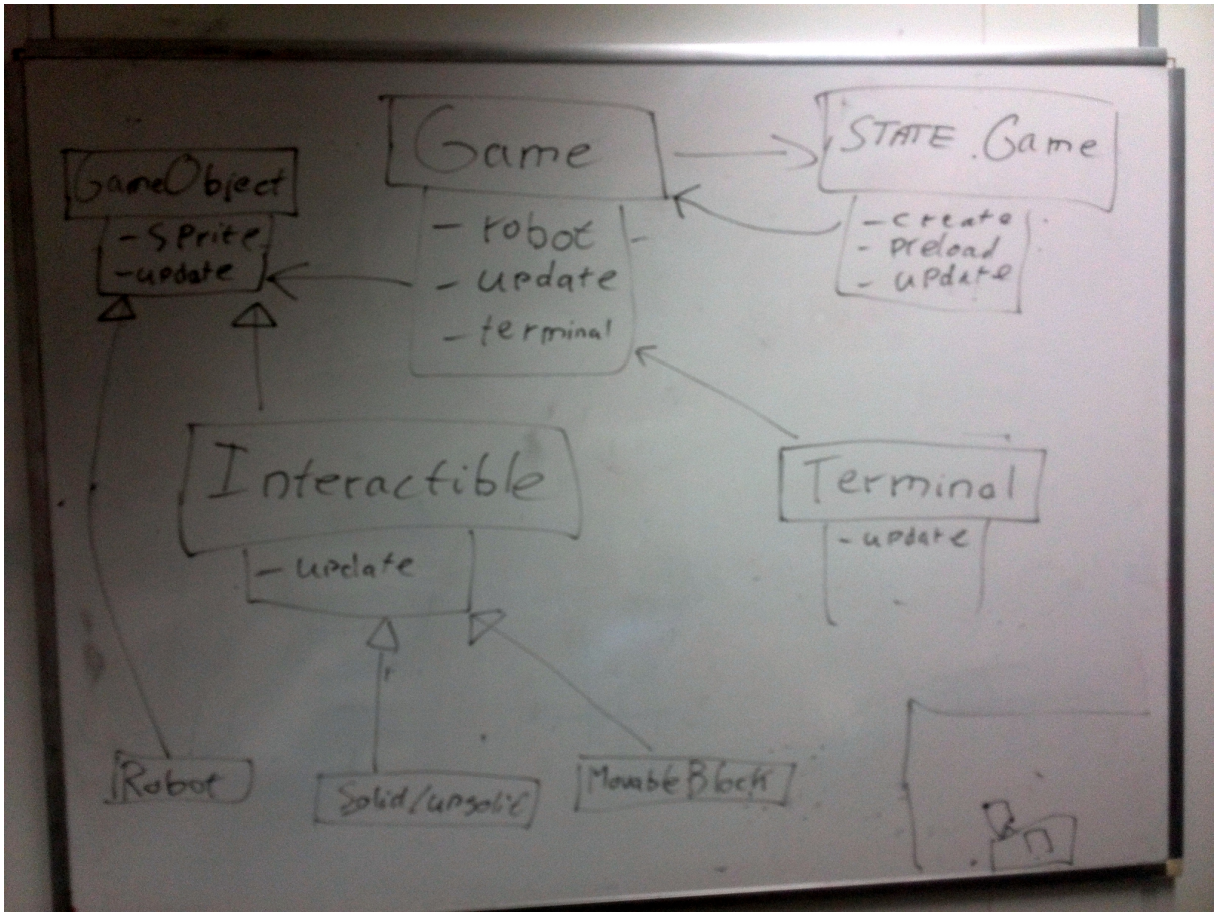

Figure 3: API structure

Many assets have been revised and updated to fit the style we opted for, different scales have been made and are readily present in the repository, allowing for easy adaptation where necessary. Others have been discarded or heavily manipulated.

We had several ideas for the Interface - first opting for floating text boxes when clicking on an object to list it's name, properties and affected methods. We opted for a terminal help function instead to avoid cluttering the game with boxes and avoid having many clicks. The focus is to 'write/code' your way trough the game, not click. Further we discussed some clarity changes - where each specific manipulation such as solid/unsolid() or move() would be visually queued by having those objects appear in a certain color only. We looked to color-code parts of the terminal input to make it clearer for the player, ultimately we had to scrap that in favour of a more robust terminal.

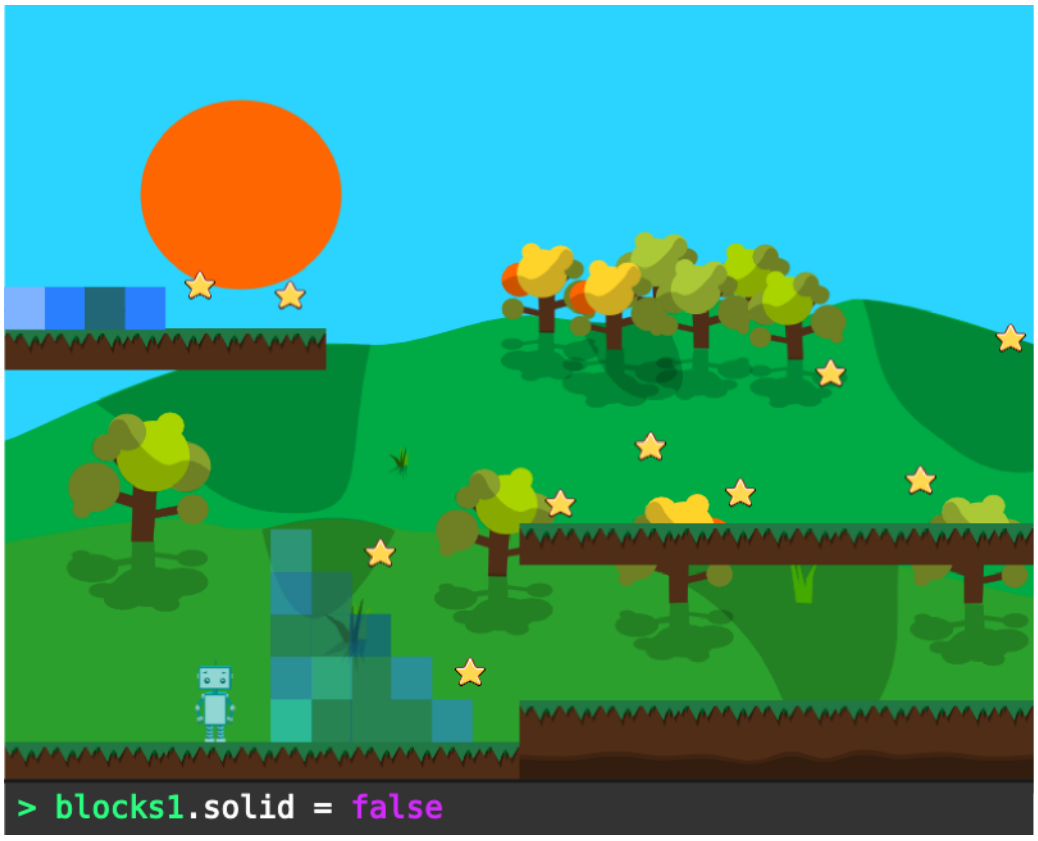

Figure 4: terminal

Our initial level was made up of a single defined area, later we introduced a camera motion that allows us to make bigger levels that scrolls with the player. This allowed for more complex and larger levels. The story progression should have a logical ordering for the levels, creating a factory level where you first become acquainted with Jumper, traversing the factory to escape into the wide world. These world locations should follow some logical sequence. We wanted to make the areas diverse and interesting so the idea was to make a neon-city progression, where you would visit the outskirts with defect robots, the inner city full of life at night-time and visit the center with a mall and disco. We instead decided to create a progression that first explains the mechanics more so we introduced the goal 'get to the TU/e' for Jumper, where would learn more about his abilities.

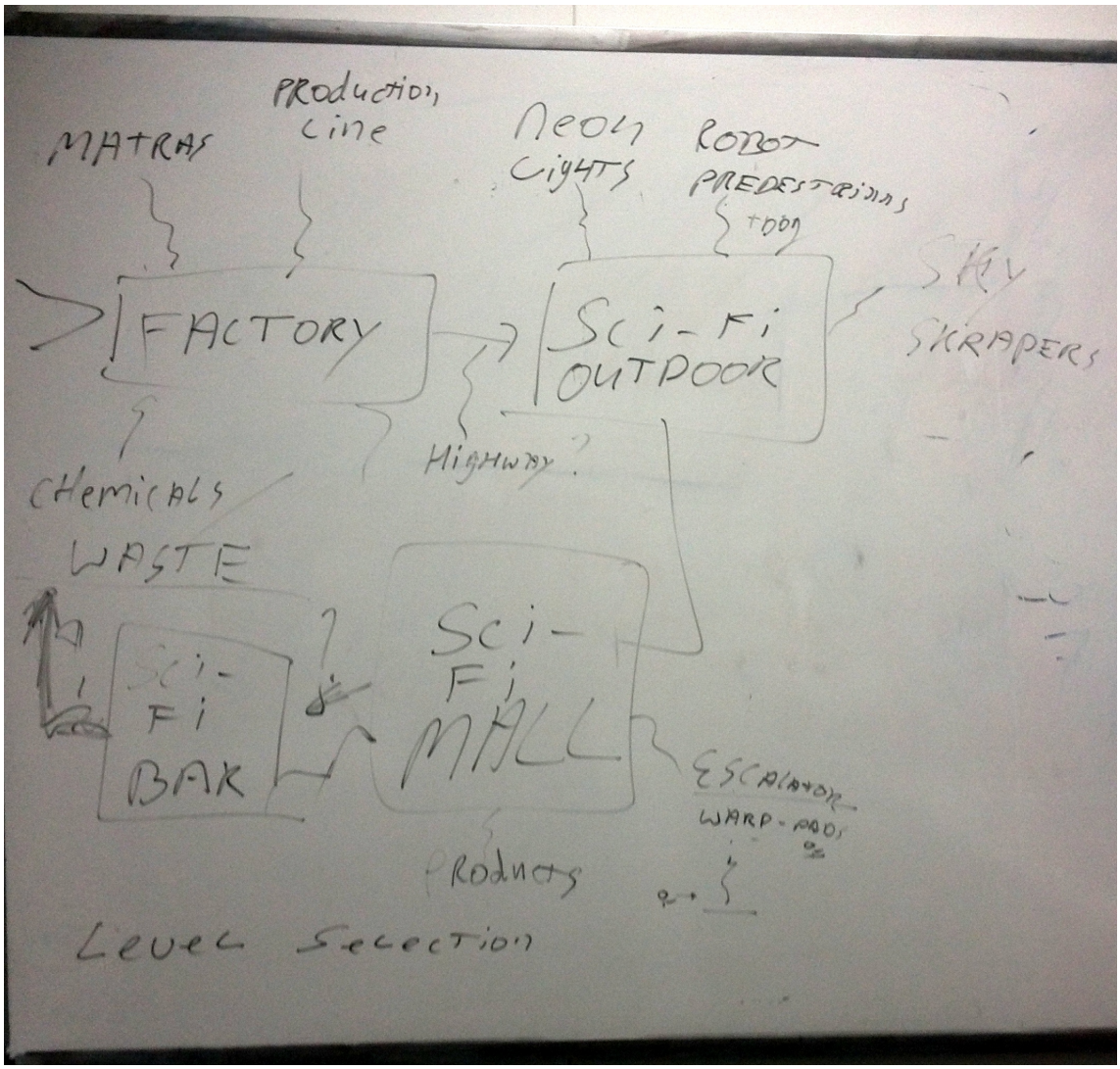

Figure 5: level progression

5 Evaluation

5.1 Individual report - members motivations and reflections

5.1.1 Kevin Cleijne

I enjoyed the course as a whole but felt that we had too similair backgrounds to really structure the roles. The game we made was a puzzle platformer which does not require a storyline as much as something like a point-and-click, which was my previous role. I decided to work on art assets and help out Jeroen but it proved difficult to adhere to the standards of flat design, as I had no experience with vector drawing. During the project I tried to help out in different areas, mostly making basic assets and doing the documentation of the design principles for Google Drive and the final report. Looking back I feel that I should have lead the project more, allowing proper focus and clear, crisp ideas/objectives for us. A lot of work proved unusable because of communication errors and unacknowledged requirements for assets that I made, requiring heavy modification or even removal of them. For the next project we really need to have a crystal clear communication line where everyone knows exactly what to expect from another. I will

need to have a look at what is at our disposal with GitHub, see if we can get these requirements clearly visible. We need to set deadlines and expectations for designs we wish to include so that we have plenty of time to incorporate it properly. I need to communicate better and ask exactly what is required of me so that the effort is not lost, which is a shame.

5.1.2 Jeroen van Hoof

5.1.3 Pieter Kokx

5.1.4 Joep Klein Teeselink

5.1.5 Twan van Schijndel

6 Appendix