PROJECT REPORT

Research and Development project

May & June 2019

Group 26:

Jelle Medendorp	*s1027748*
Janik Kreuning	*s1027402*
Axel van Abkoude	*s1021909*
Rick ten Tije	*s1005826*
Thijmen Schoonbeek	*s1027777*
Sven van der Post	*s1028679*

1. Introduction

We have made a two dimensional platformer game. We made it with the theme of our first year of studying computing science, so it will be extra fun for computing science students. But of course any other person can still enjoy our game. The targeted audience is not strict, it's just for people who would enjoy to play the game for fun. The goal in our game is to get all the EC's before the time runs out (the deadline).

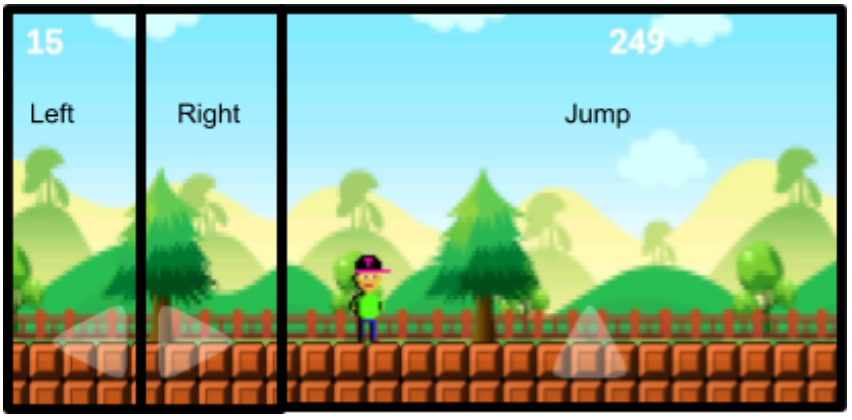
2. Description

Focus on properties

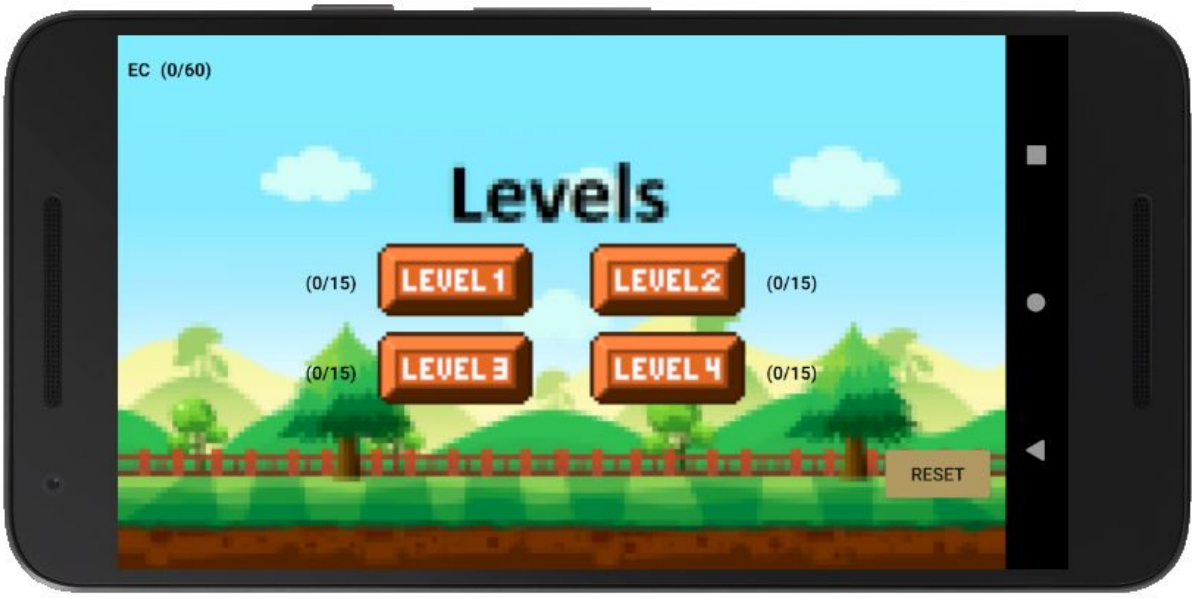
Our product is a two dimensional platformer based around the first year of the study. It is meant to be a quick to learn game with simple controls and gameplay which is familiar to any platformer which makes it a game for everyone.

The name of our app is “Collage Invaders”

The controls consist of three triangles, in the left corner there are two triangles pointing left and right which control the direction of the character and in the right corner there is a triangle pointing up which makes your character jump. The buttons for these actions are a lot larger than the triangles which makes controlling the character easier.



Picture 1: controls on screen



Picture 2: level select screen

The menu of our game is fairly simple, when you start the game you first enter the title screen which displays the name of our app: “Collage Invaders”. When you click anywhere on the screen you go to the next menu, the level select screen. Here you can select the level you want to play or reset your current progress which can be seen in the left top corner of the screen. At any point you can reset your progress and set your total ec back to 0 with this button. This also resets the levels. When you finish a level you return to the level select screen.

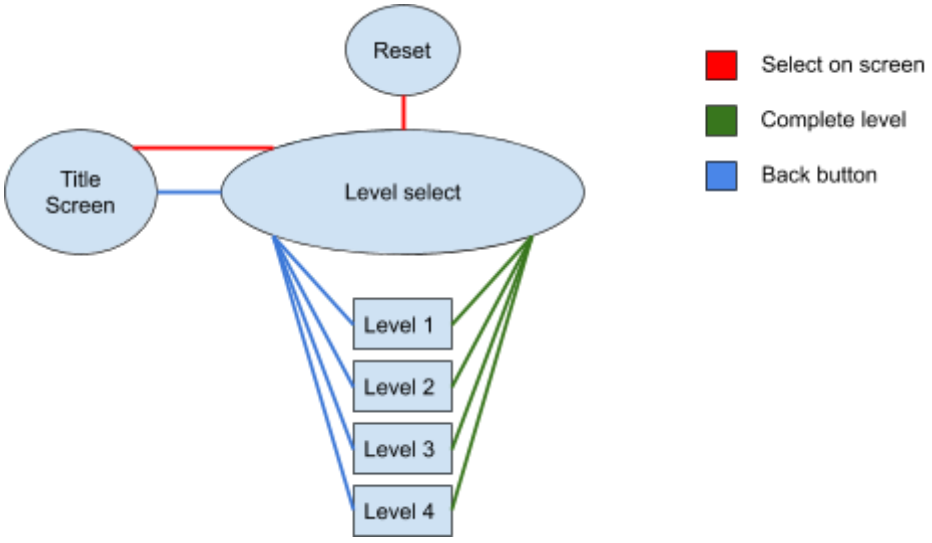


Diagram 1: menu flow

The game is divided into four levels, like the four quarters of the school year. In each level you can gather 15 EC by collecting coins which each give you 3 EC. The goal is to complete all four levels and accommodate 60 EC, enough to pass the first year!

Product justification

The reason to make an app like this one is purely for entertainment. There are a ton of apps/games that are very similar to the one we have built. Those platformer games were also an inspiration to us. But we tried to make it our own to make the theme of the whole game around our first year as computing science students. This way students can easily identify with the game and our main character.

Specifications

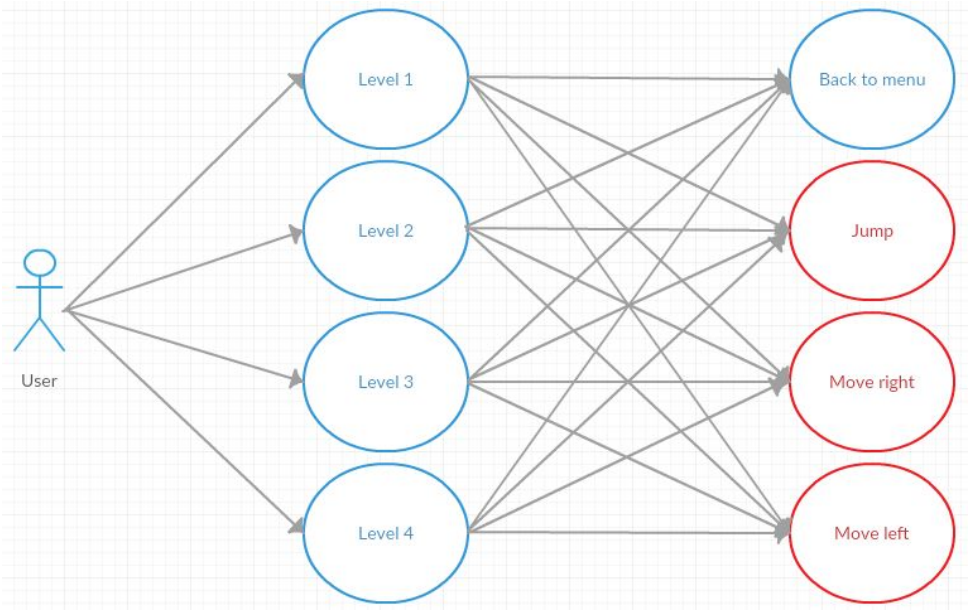


Diagram 2: user diagram

When the user opens the app. The main screen has four buttons. All for a different level. When the user chooses a level to do. There are the options to either let the main character jump, move to the right, or more to the left. The user can also go back to the main screen by pressing the back button of the phone.

3. Design

3.1 Global Design

The app was designed to be a game app which means that the flow of the system will be fairly straightforward. The MainActivity leads to the LevelSelectorActivity which leads to a LevelActivity which loads a game. To get back to the LevelSelectorActivity or the MainActivity you only have to press the back button. The data of the levels is saved through the entire application so there is no need to set a save button. We have, however, implemented a reset button to reset all the values for replayability.

This view implements our two dimensional platformer game. The gameview uses four classes to fill in the game: GroundSprite, Item, Enemy and Character. GroundSprite contains all the terrain off the levels. This terrain is only intended to be used as platforms or the ground. The Item class contains all the EC that you can pick up throughout the level. The class Enemy contains all the enemies that the character may encounter, these can differ in speed and appearance. The Character class contains the playable character. The GameView draws all of these objects to the canvas which is then displayed on the screen.

3.2 Detailed design

MainActivity

This is the main activity which will open on the startup off the app. This Activity consists of a background image and redirects you to the LevelSelectorActivity when you tap the screen.

LevelSelectorActivity

In this activity you can select which level you want to play. The Activity consists of four buttons which represent the four levels which have been implemented. These buttons all begin a LevelActivity. Furthermore the progress of each level is also displayed in the amount of ec the player has collected from each level with a total score in the top left corner.

LevelActivity

This activity loads a gameview with the implemented level. We have implemented this class to keep the functionality of the back button native to the android device. As we found that creating a separate back button was not fitting with our layout.

MainThread

With this project we needed to find some solution in order to build a frame based system, where stuff like movement and collision is possible. Since time and collision are essential parts of any game, our framework needed to at least support these systems. Therefore we implemented the main thread class that'll start and run the game and allow the gameview to update characters and collisions every frame.

GameView

Apart from the MainThread and the MainActivity, the GameView is probably the most important component of the entire game system. The game view basically creates everything players can see in the game and updates all of it every frame after its run and draw methods are called by the main thread. First of all, directly when the game view gets instantiated, it will create six important arrays of objects;

- characterSprites
- enemySprites
- itemSprites
- groundSprites
- overlaySprites
- background

These arrays hold characters, background sprites, UI buttons, etc. This way we can update all of these objects every frame the game is running.
After making all objects, the gameview will call the draw and update methods on objects that need to be updated (grounds sprites for example do not need updates, as they're static), and it will all call the moveX method whenever the player touches one of the on-screen buttons. The moveX method then passes the direction to ground & enemy sprites to trigger movement on those objects.
The way we've implemented this system, instead of moving the player character, it will move everything around the player. Since we did not know a way to build a proper camera object that moves with the player, we needed to find another method of keeping the player in view. The current implementation is a reasonable solution, as it does allow the player to always see the main character, but performance wise it might not be the best choice as instead of one object, we're moving a lot of objects at the same time.
One more thing that happens in the game view component is collision detection. Since all player movement also happens in this class, we felt it makes sense to check for collisions as well. Basically, every character and object in the game has a Rect variable, that holds the screen portion where that specific object will be drawn. We can check the left, right, top and bottom values of this Rect and compare it to other objects to check for collisions. For example, to detect player collision, we check each ground Rect and compare their left and right values to the player Rect left and right values. If the top and bottom positions also line up, we cannot move anymore because a ground object is in the way of the player.
We also implemented a timer with the deadlineStart function this function starts a timer from the default time when the time expires the thread will be stopped hereby stopping the game altogether and freezing everything in place.
Finally, in the game view we decided to pass a lower movement speed to the background than to the foreground, to create the illusion of depth.

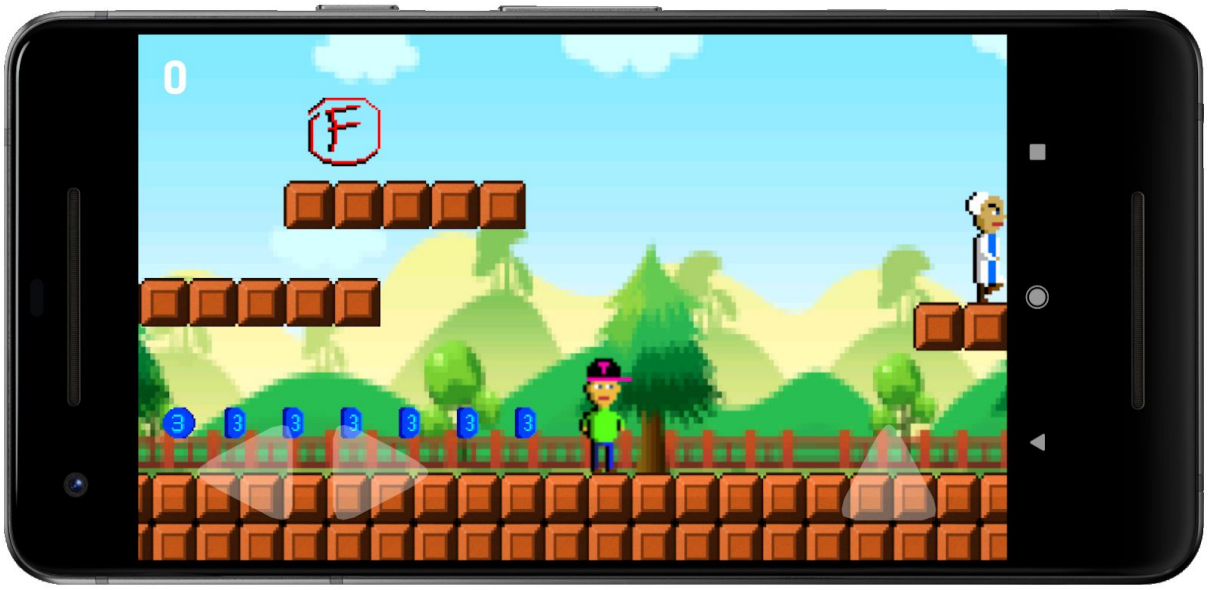
Animator & animation

These two components were inspired by existing game engines, as they allow for a structured implementation of character animations. The animator gets updated by the parent character every frame, and in this update it will increase its timer. After updating the internal timer, it updates the current animation frame using the animation component. The animation component holds a set of bitmaps, that will be shown while playing the animation, together with a name and the delay between every frame. This allows us to for example play the walk animation faster than the idle animation.

Right when a character is created, we also create an animator and pass some animations to it. We then pass that animator to the character constructor, so the character can play animations on the animator using names specified in the animation objects.

Character

The character component draws the main character and updates its animator component. It will also handle physics like gravity, using x and y velocity variables and updates the values when jumping. Apart from handling gravity, it also has a groundCheck method that returns whether or not the character is currently grounded. This way we can update walk/jump animations and check if the character can jump again.



Picture 3: Main character in the game view

GroundSprite

The ground sprite class is very similar to the character classes, except for the fact that it's much simpler. The ground view does not need to be updated, it just needs to move whenever the player wants to move. Therefore, it only has a constructor, a draw method and a moveX method.

Enemy

The enemy component is essential in any game, because it creates the main challenge for the player. Although we did not have the time to implement a game-over system when the player accidentally hits an enemy, we did implement the enemies. Enemies are simple character that walk left and right in a certain pattern, and they can be killed by jumping on their heads. While much of their functionality is a lot like the player functionality, we still faced some new challenges creating the enemy characters. Firstly, when creating a new enemy in the game view class, not only do we need to pass animations for movement, we also need to pass two animations that handle the enemy die states. Namely, when the enemy dies, it first starts the die animation, then it will start counting until 15, and finally it will stop drawing the enemy bitmaps by checking the counter value.

Also, the enemy has the added functionality to walk around. The way we've implemented this is quite easy, because we're just keeping track of a minimum and maximum x value, and invert its x velocity whenever these values are exceeded. We do need to update the min and max x values when the player walks, because as mentioned earlier, the player doesn't really move but instead we're moving the environment around the player. If we were to not update the min and max values for the enemies, these values would stay the same in screen-space, which results in the enemy moving with the camera (or so it seems).

Item

Items are small objects placed in the map for the player to interact with. Currently, the items implemented are EC coins. The player can collect these blue coins to improve their ec score and open enter the next level. Just like any other component, coins work with a Rect to keep track of their position and size, and they only have one animation. Coins always rotate, and when the player interacts with them and picks up a coin, the coin will not be drawn anymove so it seems to have disappeared.

Background

Backgrounds are really quite simple, and maybe we could have even left out the background class entirely. The background works the same as the ground sprite class, it just moves slower to create a three dimensional illusion, and it is not passed to the character to make sure the player character does not interact with the background. We did pay close attention to the drawing order, to make sure the background images are drawn behind the environment objects, and not in front (which would make everything disappear).

OverlaySprite

The overlay sprite is mostly just decoration. It gives the player an indication as to where to touch the screen to move the player around and jump. The buttons themselves don't actually do anything, it's just a bitmap drawn in a certain Rect.

NumberLabel

The NumberLabel class is a class to convert a number into a canvas drawing. The class transfers integers two digits long into two drawable numbers and integers one digit long into one drawable number. There was no need to create a third number because it was never the plan to make a level with more than 99 EC.

TimerLabel
The TimerLabel class is a class to convert a number smaller than four digits into a canvas drawing. The class makes it possible to draw integers on a canvas. Because we did not intend to make the time to complete a level above 999 seconds we have not implemented the fourth number. Because our first intention was to make a time layout with a colon this had become its own class so in future development this could be implemented.

PrefManager
The PrefManager class saves data to a SharedPreference which normally stores the preferences of an app. This class can also be used to store limited amounts of data. Because our app does not need to save a lot of data this was the perfect solution for us as it also saved through updates or shutdowns. This way the progress was never lost.
The SharedPreferences can not be edited directly that is why you need to implement an editor where you can write data to with a key and later apply this to the preference file of the app. When you want to get this data you simply get the value with a get method and a key.

We have implemented all the EC as boolean values this way we could track which EC has been collected and thus which EC had to be shown and collectable in a level. Because of the way the SharedPreferences work you can not put an array of booleans that is why we implemented an array by concatenating the index to the “ECarray” key.
To get the EC we have created two methods: one to get the amount of EC in the array from a certain index to another, this was for setting the amount of EC collected in the LevelSelectorActivity and another to get the value of a specific EC this way we could check if a certain EC had to be collectable or not.

To keep the deadline the same value when you would pause the app while playing a game the deadlineSecond has also been implemented to be saved to the PrefManager.

3.3 Design justification
The way we’ve structured the class hierarchy allows for further development and improvements. We decided to go with the game view that functions as the main component of the game holding everything together. By keeping the Activities and the GameView separate we make the application easily expandable because we can simply add new items characters and ground objects to the GameView. We might have been able to further improve the system structure by checking for code duplication and unnecessary classes. An example here would be the ground sprite class and the background class. The only difference is that we pass the ground sprite objects to the player, and we’re moving the ground sprites with a different speed compared to the background. We could have easily used a single class for these elements. Furthermore we could have implemented Levels with its own class but due to issues with returning the object arrays through classes we were not able to implement this. This was also the case with using Lists instead of fixed size arrays for the levels.

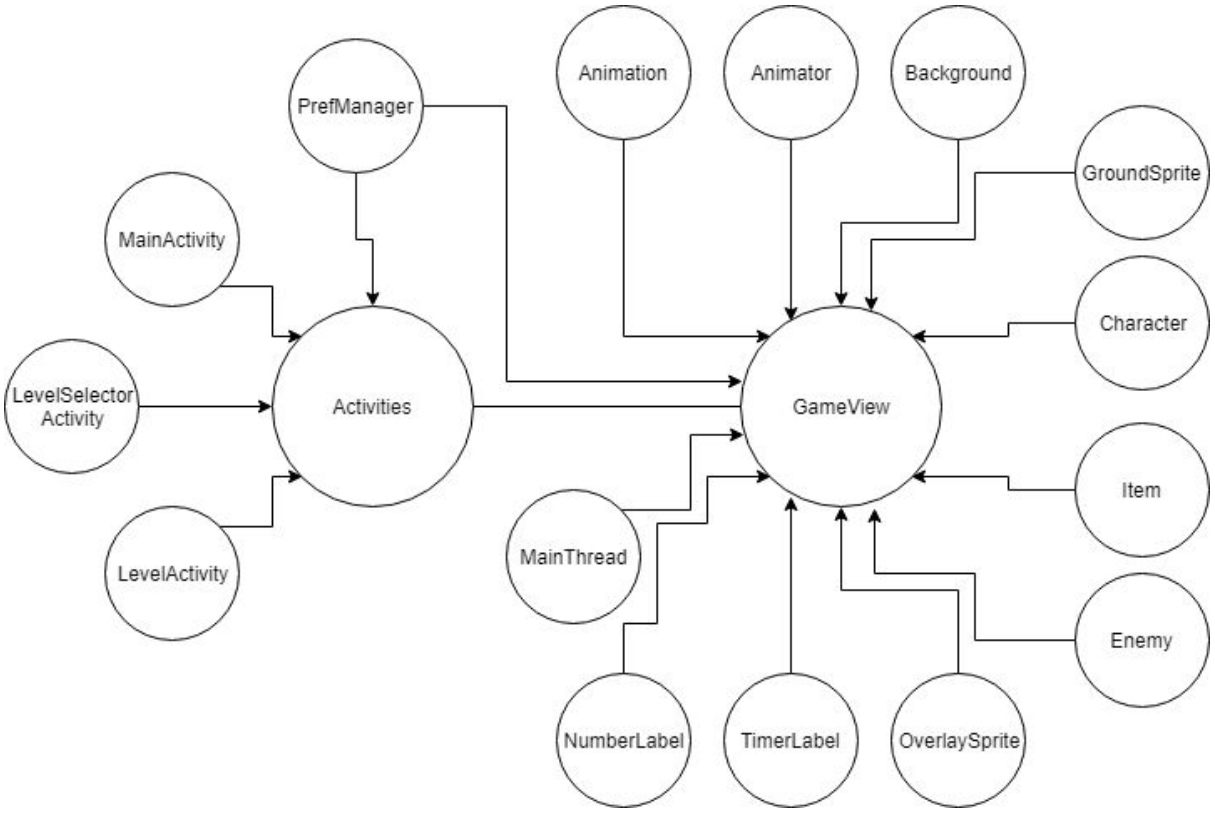


Diagram 3: Design choices

As you can see in the diagram above we have designed the app with the idea to keep the activities and the game as separate as possible, the only classes who will break these design choices are classes which store global variables. In our case this will be the PrefManager.

4. Project Management

To create our app, we tried to divide the tasks between the team. Although with six members, it was quite hard to do so. At the start we divided the project in the following teams:

- Level builders
- Game engine design
- Graphic design

The level builders were responsible for creating the levels of our game. In the beginning we had way more features in mind for the game so the level builders made more interesting levels than we put in our final app. This meant that most of the levels they made were altered and simplified due to reasons stated in the evaluation. The level builders were Hugo and Jelle but after Hugo left, Sven joined the level builders team.

The graphic design team was responsible for making the graphics in the app. This includes moving animations for the characters and items in the game. All of the graphics were made by the graphic design team with the exception of the background, the background was found on the internet and the graphic design team pixelated the art. We decided to make the art in the game pixelated since we thought that looked nice and also fun to work with. The graphic design was done by Rick.

The game engine designers are responsible for the top level game components like the character controller, enemy class, collision, gravity, etc. They make sure everything works together nicely and the player can walk around and interact with the level. Initially, we planned on having three game engine designers, as this is probably the largest, most time consuming portion of the development process. We then quickly switched to two engine designers, because we found out Rick's talent for graphic design. Even though in the first stage we wanted Thijmen en Janik to do the engine design, most of the engine was created by Axel and Thijmen eventually. Axel especially did a lot when it comes to the system backend, like saving systems, menus and bringing the whole game together. Thijmen on the other hand did most of the visible systems like character physics, enemies, items and backgrounds.

Week	Game Engine	Graphic Design	Level Design	Brainstorming
1				
2				
3				
4				
5				
Legend:	Active			
	Idle			
	Inactive			

Table 1: Weekly tasks

Every week on Thursday, we had a project meeting in which we discussed possible improvements to the app and gave the other teams tips on what could be better. We also worked on the game during those meetings if we had time left. These meetings took around two hours and at the end of every meeting we discussed what everyone needed to do at home for that week. Not every meeting was as efficient as the others but most of them went well. The main problem during the planning was the leaving of our dear group member Hugo which shook all of us tremendously. Due to his leave, we had to postpone his work a bit until Sven joined the team and could pickup from where Hugo left.

When creating the graphics, Rick sent the graphics to Thijmen to test if the animations and other art worked well within the app and Thijmen would give feedback to Rick when some things should be changed, like resolution. When the level builders had an idea, they would contact Axel and ask if it was possible or if he could implement a level they had just created.

5. Evaluation

When we started our project, we thought we could create a good looking platform genre game in android studio within five weeks. However, during those five weeks we encountered some problems regarding Android Studio. We came to the conclusion that android studio was not suited for making the kind of game we had in mind even though it was suited for really simple games like Sokoban. But we decided to finish the platformer nonetheless since we didn't have time to change to another project idea.

After the second week of the project one of our team members, Hugo Ulfman, left which was not exactly beneficial to our project. However, he did manage to arrange a replacement rather fast so we had a team of six again.

Even though we had some problems in the first two weeks of our project, we did manage to make a working, decent looking platformer game. At the start of our project, we had a lot more features in mind for the game like a currency, more levels, more different enemies, character selection and much more. During the project it became clear that we neither had enough time, nor was Android Studio suited for making such a complex game.

When we tried to implement all the features that we wanted in the beginning we found out that Android Studio does not have features that would help a lot when creating a game. Features like built-in elements such object transformers and animators. We thought Android studio would have these features since other engines like Unity do have them but we were wrong. We had to make all the animations by ourselves due to the absent of the animator which took some time. Another downside to making a game in Android Studio is that Android Studio does not scale apps automatically on different mobile devices. To circumvent this we all needed to test it on the Nexus 5X API 28 emulated device in the AndroidStudio IDE.

In the end, we are satisfied with the product and we learned a lot in these last five weeks. We learned how to code in Java, even though we had already a little knowledge about Java form the Object Orientation course and the Sokoban app, and we learned to organize and work together. In the beginning we had some troubles with organizing and communication with each other but that went better after a few weeks.

In the future we would change quite some things when working on an Android Studio project. First of all, we would choose another type of program to make in Android Studio since making a game is not optimal in our experience. Secondly, we would start earlier with the project and divide the tasks better, we already improved our planning through the last five weeks but if we started with a good planning, the product would have been much better. Last but not least, we would try to find a better way to share our progress with each other such as code and other relative components in our project. Copying the code and sending it to each other was not really efficient so we probably would use GitHub or another program to easily share and work on the same project at the same time.