AALBORG UNIVERSITY
STUDENT REPORT

Department of Architecture,
Design and Media Technology
Medialogy 10th semester

Title:

Using Tangible Widgets for Travel in a Touch-Based Tablet Game with a Scrolling Camera

Theme:

Masters Thesis

Project period:

February 2016 — May 2016

Project group:

MTA 161036

Members:

Mathias K. Berthelsen

Gustav Dahl

Benjamin N. Overgaard

Supervisor:

Martin Kraus

Abstract

Tangible widgets have been used in the context of touch-based games for children, where widgets represent game avatars that players control with the widget. Due to a lack of prior research in this field, this thesis investigates problems based on an initial study where children play the game *Disney AppMATes*, which uses a scrolling camera. In this game, players control a virtual car using a widget. In the initial study, it was found that children had a desire to drive on roads. However, roads were often difficult to drive on because of the view of the scrolling camera. This thesis proposes a range of solutions to the problems found in the initial study, but focuses on the above-mentioned problem for which a solution called *Road Focus* was designed. Through an experiment with 64 participants in ages 5-7, it was found that *Road Focus* is a significant improvement to the solution used in *AppMATes*.

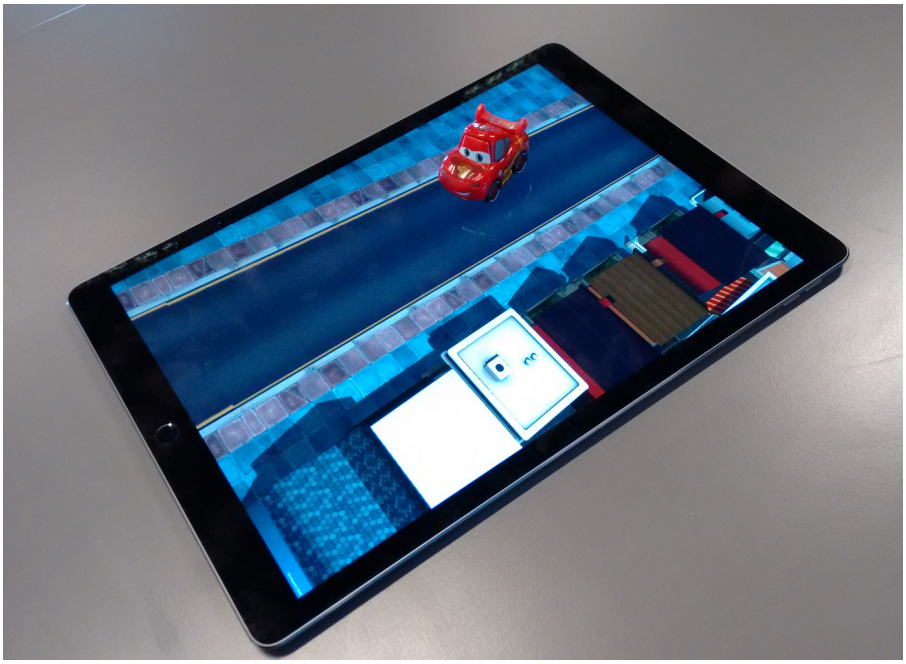
Aalborg University

Medialogy 2016

Department of Architecture, Design and Media Technology

Using Tangible Widgets for Travel in a Touch-Based Tablet Game with a Scrolling Camera

Mathias K. Berthelsen, Gustav Dahl, Benjamin N. Overgaard



Abstract

Tangible widgets have been used in the context of touch-based games for children, where widgets represent game avatars that players control with the widget. Due to a lack of prior research in this field, this thesis investigates problems based on an initial study where children play the game *Disney AppMATes*, which uses a scrolling camera. In this game, players control a virtual car using a widget. In the initial study, it was found that children had a desire to drive on roads. However, roads were often difficult to drive on because of the view of the scrolling camera. This thesis proposes a range of solutions to the problems found in the initial study, but focuses on the above-mentioned problem for which a solution called *Road Focus* was designed. Through an experiment with 64 participants in ages 5-7, it was found that *Road Focus* is a significant improvement to the solution used in *AppMATes*.

Contents

Chapter 1

Introduction

The concept of merging the real world with the virtual world has been of interest for researchers as early as the mid-1990's [FIB95]. Tangible user interfaces are broadly concerned with giving physical form to digital information. Manipulation of these physical objects are used to interactively engage with computational systems [Ull02].

In essence, it's the idea of blending physical and virtual artifacts. Researchers have used many different terms to discuss this area, such as *graspable user interfaces* [FIB95], *natural user interfaces* [WW11], *tangible user interfaces* [Ish07, BCL12], *mimetic interfaces* [Juu10] and *tangible widgets* [BFTK14]. [HB06] use *tangible interaction* as an umbrella term. For this project, we will use the terms *tangible interactions* and *tangible widgets*

According to [HB06], in relation to tangible interactions, most frameworks take a structural approach that systematically map out an abstract design space, but they seldom address the human interaction experience. However, it is widely agreed that by moving human-computer interaction from the virtuality of the screen into the physicality of the real world, the design space is significantly extended and thus enables new and rich forms of interaction [HSU04].

Tangible interactions have in recent years been used in the context of console and tablet games with commercial products such as *Skylanders*, *Disney AppMATes*, *LEGO App Brick* and *Fabulous Beasts*. Although the target group for such products primiarly consists of children, many studies have been conducted on adults [BFTK14, BFTK15, KWRE11, BCL12]. Furthermore, these studies focus on the affordance differences between using physical widgets on a touch

screen and using fingers on a touch screen. To our knowledge, there are no studies that focus on solving the actual problems that occur in current games that use tangible widgets.

For instance, [BFTK14] compare controlling a tablet game with finger touch to controlling it with tangible widgets. They found that participants preferred using tangible widgets due to ease of control for most interactions in the game. However, they also found that the widget could occlude parts of the screen. Although they mention that it would be interesting to test tangible widgets on children because of their similarity to regular toys, none of the test participants in the study were children.

When exploring the different tangible widget games commercially available, we found that *AppMATes* was the only example where the game world is bigger than what can physically be displayed on the screen, which resulted in it using a scrolling camera. We find scrolling cameras interesting and relevant, since they provide more freedom to design virtual worlds bigger than what can fit on a physical screen. Based on a few internal tests, we found a group of interaction problems with *AppMates*, which will be described later. Furthermore, using a scrolling camera in a tangible widget game has, to our knowledge, not been explored in previous studies.

In order to further investigate the problems found in *AppMATes*, we conducted initial tests with children playing the game. These will be described in the next chapter.

Chapter 2

Initial Observations of Children using AppMATes

Disney AppMATes is based on the Disney/Pixar movie *Cars 2* (see Figure 2.1). In the game, the player can freely drive around in a top-down view and take on different missions. The game starts in free roaming, and stays so for the better part of the game, but also has specific tasks such as collecting objects, racing other cars and playing minigames. The game allows for two ways to play: with a physical widget, which represents a car, or with touch (with a virtual representation of the widget controlled with two fingers). For this project, we mainly focus on the free-roaming interactions where a widget is involved. Through this report we use the words *widget*, *physical widget* and *physical car* interchangeably.

Figure 2.1: Tangible widgets are used to play *Disney AppMATes* [Hyb13].

Players control the in-game camera in *AppMATes* by placing the widget on the screen while

touching its conductive parts on the front and side windows. As long as the widget touches the screen and the fingers touch the windows, the player, and thereby the virtual camera, moves forward. The movement of the game camera is dependent on the widget's orientation and the car's acceleration. Moving the widget has no influence on the game camera, but is used to move around freely inside the boundaries of the screen and interact with objects.

In order to understand how children naturally interact with widget-based games in the style of *AppMATes*, we conducted an informal test with a group of children. The objective was to discover potential issues and pitfalls that could be used as a starting point for further investigation.

2.0.1 Procedure

We let seven children with an average age of 6.4 years ($SD = 2.2$) and five adults play *Disney AppMATes*. All sessions were video recorded for further analysis. The children and their parents were approached at a local library, whereas the adults consisted of students from various departments.

Our testing approach was inspired by [DR04, DM02]. They found that, without prompting the children, it is not guaranteed that the they will actually speak at all. Therefore, we used intervention and talk-aloud methods to make the children give feedback on their experience. The talk-aloud approach aims to let the children talk about what they are *doing* and not so much what they are *thinking* (as in think-aloud). We further used it to find out if they understood what they were doing and what the game expected of them.

Each play session lasted 8-15 minutes. The players were asked to drive around freely for the majority of the time. When the facilitator deemed that they understood the game to a satisfying degree, the participants were prompted to try out different aspects of the game, e.g., by asking them to drive to a specific location or complete an in-game task. The facilitator also asked them to experiment with the tangible widget, such as lifting it, moving it faster and replacing it with another widget.

2.0.2 Common Issues Found

During these tests, we found that both children and adults had difficulties with several aspects of navigation. In parts of the game where they wanted to drive faster, they pushed the widget forward, resulting in them eventually pushing the physical car to the very edge of the physical screen, effectively making it difficult to see what was coming in front of or next to them in the virtual world. This often caused them to collide with objects in the environment which they otherwise would have avoided. Even when they seemed to realize pushing the car didn't make it go faster, both groups still kept doing it.

We also found that when players rotated the car, they often lost their grip of the widget and accidentally let go of its conductive parts. This caused the widget to lose connection and the car would stop, thereby frustrating the players. Furthermore, rotating the car caused problems with the physical limitations of their hand: their hand and arm occluded the screen, and awkward positions forced them to temporarily let go of the widget.

As the game tries to map the widget directly to the car in terms of size, in scenarios where the car is driving quickly, the camera can't zoom out and show what's coming ahead, resulting in players having difficulties reacting to obstacles.

A large part of the players in both groups had a desire to follow the in-game roads. Sometimes, while driving alongside the edge of the screen, they would end up driving a bit skewed towards the center of the screen compared to the road. This meant they would drive slightly away from the road, making it gradually less visible. In the end, this would force them off the road. Some players kept driving off-road, while others lifted the car, thought for a bit, before putting the car back down next to the road and driving back up on it. In both cases, it seemed to annoy the players.

Likewise, we saw several times where what seemed like their mental model of the in-game universe appeared to keep them from doing certain actions. Every player, even the most energetic, tried to avoid collisions, even though the virtual in-game collision does close to nothing. However, when the game showed an arrow to indicate where to go, they drove directly in the direction it was pointing and completely ignored any potential collisions.

On the same note, they almost never lifted the car and placed it somewhere else on the screen

than where they initially picked it up. And if they did, it was always close to where they lifted it, and never on top of a wall or obstacle. This became a problem in two instances:

1) To enter a menu, players had to drive to the corresponding spot in the world. When the menu opened, the widget would occlude it, which generally made the participants remove the widget from the screen. Nevertheless, when they exited the menu, they would place the car back into the spot from where they had just lifted it, meaning they would open the menu again. For some participants, this even happened repeatedly and caused a great deal of confusion.

2) If participants ended up at the edge of the game world, where they couldn't travel any further, the game expected them to turn the widget around and drive back. Sometimes they would have so little "wiggle room" between the edge of the screen and the edge of the game world, that they had a hard time placing the widget. However, this was only their own perception, as they could easily just put it directly on, or beyond, the edge of the game world and drive back. Each participant was shown that lifting and moving the widget did nothing, yet the problems kept occurring. The game itself even prompted the player to place the widget back on the middle of the screen when pushing it off the edge (see Figure 2.2). However, we noticed that many participants simply ignored this.

Many participants felt they didn't know what they were supposed to do in the game and only coincidentally encountered activities. We assume this was largely because the participants couldn't see very far and therefore had no apparent goal to pursue. The game features a pop-up map, which every participant could successfully read and understand, but when asked to drive to a specific spot shown on the map, they had to open the map several times during their travels to find it. Again, we assume this is because they couldn't see very far and would therefore miss out on visual cues in the game world.

When asked about how to accelerate or decelerate, most participants didn't know how to control it. This was a trick question, since it isn't possible to control the acceleration in *AppMATes*, aside from lifting the widget (but then the car isn't technically present in the world anymore). Some assumed they had to either press the car down harder or push it closer to the edge of the screen in the direction they wanted it to go. Participants tested it out and realized it did nothing. In the same regard, we asked them how to stop the car, to which everyone just let go of the widget and assumed this meant the car had stopped. The fact is, the widget isn't being tracked anymore, which means letting go of the widget is the same as lifting it.

Figure 2.2: If the player moves the widget too close to the screen edge (indicated by two white squares in the bottom), *AppMATes* shows a prompt to move it back to the center.

2.0.3 Main Issues Found in AppMATes

For this project, we primarily focus on one found during the initial observations with the *AppMATes* game. The defining aspect of widget-based games is the interaction between the physical and virtual world. As described above, several participants had problems with continuously pushing the physical widget to the physical boundaries of the iPad screen. This was a major problem, since it prevents them from seeing what is located in front of them or to their side (because the screen physically cannot display more of the virtual world) and also what is behind them (because their arm obscures the screen). Furthermore, they need to somehow consciously re-position the widget in the center of the screen. We decided that the goal of this project would be to design interaction methods that motivates the children to *not* position the widget close to the screen edges. Even Disney seems to acknowledge the problem, as shown in Figure 2.3.

Figure 2.3: One of the help menus in *AppMATes* describes how to hold and place the tangible widget.

Chapter 3

Background

In this chapter, we explore areas related to the problems found in the initial observations of children playing *AppMATes*. This chapter investigates a potential age group to test our design with; important aspects of tangible interactions; travel in virtual environments; and virtual cameras in relation to digital games.

3.1 Choosing an Age Group

To specify the age group to focus on when designing and testing potential solutions, we have used knowledge gained from the initial tests, as well as research within this area. This will be described in the following.

Critical transformations occur in the early years of childhood. Even small age differences can have an impact on the child's social, sensory-motor and cognitive skills [FB06]. Therefore, it is important to find an appropriate age group. The recommended age for *AppMATes* is 4 years and up [1].

There are different theories on child development stages. *Milestones* define recognised patterns of development that children are generally expected to follow (obviously, there will be individual differences from child to child). One of these is Piaget's theory about cognitive development [Hou08]. He proposed that children go through a series of stages in their development in order to obtain logical, analytical and scientific thinking. Piaget proposed four development stages:

[1] http://www.amazon.com/Spin-Master-20051780-AppMATes-McQueen/dp/B005GJSRAK/ref=sr_1_1?ie=UTF8&qid=1461070218&sr=8-1&keywords=appmates

- The sensory-motor stage (0-2-year olds)
- The preoperational stage (2-7-year olds)
- The concrete operations stage (7-11-year olds)
- The formal operations stage (11-16-year olds)

In the following, the first and last stages will not be described, since we have deemed them outside the age group for a game like *AppMATes*.

Preoperational children are characterized as being egocentric, meaning they see the world from their own perspective and have a hard time experiencing it from someone else's point of view. They also tend to only be able to concentrate on one characteristic of an object at a time. This is important when using interfaces that require navigation: generally, navigation through hierarchies should be avoided for this age group [Hou08].

Children in the concrete operations stage tend to be better at appreciating someone else's perspective. Likewise, they are able to understand hierarchies, which enables them to use a greater variety of technologies [Hou08].

Younger children have a hard time playing games with rules, even if assisted by adults. However, by the age of 4, children begin to be able to play small games with simple rules that depend on chance but not skill or strategy [FB06]. At age 6 or 7, games should still remain simple and straightforward with only a few rules and not requiring too much skill. When children reach the concrete operations stage at age 7 or 8, they become more capable of formulating and carrying out plans, meaning they can enjoy a wider range of games with more sophisticated rules and themes [FB06].

According to Piaget, most games can be classified into one of four groups (note that he was talking about non-digital games). These correspond well to the four development stages mentioned above: games of practice (age 0-2), symbolic games (age 2-7), rule-governed games (age 7-11) and games of construction (age 11-16) [FB06].

Looking at exploration tasks, where children manipulated objects without being able to see them, Piaget and Inhelder identified three stages [Pow99]. These are as follows:

- Stage 1: Child engages passively and randomly (3-4 year olds)
- Stage 2: Child is more active in the exploration, but not always systematic (4-6 year olds)

- Stage 3: Child explores following a systematic, general plan (6-7 year olds)

Based on our initial tests with *AppMATes* with children aged 3-10, it was clear that children at age 3-4 were too young to understand the game. They lacked a general sense of direction and purpose. On the other hand, it was evident that children aged 9-10 quickly became bored by the game; it was just too simple for them (many noted that their younger siblings might like it, but that they themselves were too old for a game like this). Taking inspiration from the theories above, we have chosen the age group of years 5-7. Children these ages seemed to be engaged in the game world in an active way, which means they want to explore it. They also understand games with simple rules. It is important that they have reached the state where they can understand symbolic games, i.e., being able to use objects that can represent something else than its original function. Even though tangible widgets often strive to look like what they symbolize (e.g., a car), the child needs to understand that the widget also is a representation of the player avatar.

3.2 Tangible Interactions

In the following subsections, we will focus on important aspects of tangible interactions and relate them to *AppMATes*.

3.2.1 Physical Affordances and Cognitive Load

According to [FIB95], the affordances of physical artifacts are inherently richer than the affordances of virtual artifacts. When talking about *graspable user interfaces*, key advantages are mentioned, such as facilitating more direct interactions that make use of keen spatial reasoning skills.

Similarly, [Che03, Nor13] state that the affordances of real-life physical objects help users to construct coherent mental models [Cou15]. These models, however, aren't necessarily applicable to the virtual world. Therefore, it is key that designers understand the affordances and interactions of both to ensure that they do not cause confusion to the user [Cou15]. The user might understand the software constraints and feedback, but it may impose a small cognitive load. Using a physical input device — a tangible widget — with physical constraints matching (or coming close to) the virtual constraints can remove this load [HPGK94].

In the case of the *AppMATes* game, the player can move the widget around the screen, but the implied benefit of this is that the player actually speeds up or slows down the car. The problem here is that the camera does not follow this logic, and therefore the player ends up pushing the widget to the edge of screen and is unable to see what lies ahead.

3.2.2 Coherence Continuum

[KBNR03] introduce the concept *degrees of coherence* as a means of distinguishing between different types of tangible user interfaces. The framework consists of the concept of links between the physical and digital objects. How closely they are linked together is described by a set of underlying properties. It is proposed that the relationship between physical and digital objects can be rated along a coherence continuum (see Figure 3.1). Each level represents the extent to which the linked physical and digital object might be the same thing. For example, whether the physical and digital objects are seen as one common object that exists both physically and digitally, or if they are perceived to be separate but temporarily interlinked objects. The weakest levels in this continuum are deemed to be computational artifacts that operate as general-purpose tools. The strongest levels are proxy projections, which encompass a relationship where the digital artifact is seen as a direct representation of some of the properties of the physical object.

With *AppMATes*, we perceive the physical car as being closely related to the virtual car. This became increasingly apparent when test participants let go of the physical car and thought the virtual car had stopped, when in fact the virtual car doesn't even exist when the physical car isn't touched by the user's fingers.

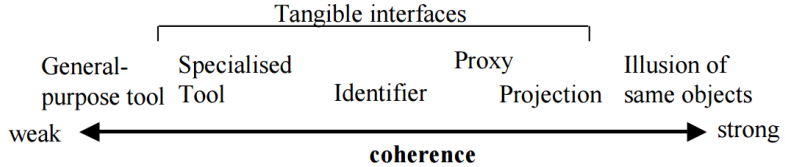


Figure 3.1: [KBNR03] proposed a framework to describe the relation between physical and digital objects.

[HTP+97] found that users perform better when the input device has a generic shape, in contrast to if the device replicates the shape of the virtual object that is being controlled. However, [BKLJP04], who use the term *physical prop* for input devices that replicate virtual objects,

state that such devices might be beneficial when the application focuses on aspects such as speed of learning, sense of immersion or user enjoyment.

We are mainly interested in using the widgets as entertainment for children, so the focus will be on making the game as intuitive and engaging as possible. Precision and task performance are thus less important for this project.

With *Disney AppMATes*, it seems like the goal is to reach the right side of the continuum, i.e., to make a close link between the physical car and the virtual car. For this project, we have decided to follow the same goal to let players feel like they are controlling an avatar, which in this case is Lightning McQueen from the movie *Cars*.

3.2.3 Expectations

An input device might change multiple quantities (e.g., *x, y, size*) [HPGK94]. The user's perception of the interdependencies and relationships between these quantities should mirror those of the input device. For instance, users found a 3D tracker more efficient for changing (*x, y, size*), because the quantities felt closely related, whilst a mouse felt more efficient for changing (*x, y, greyscale*).

Players are unable to control the acceleration of the car in the *AppMATes* game, but during the initial test it was observed that they initially believed this can be controlled through the widget somehow, either by pushing it forward or pressing it down. This lead us to believe that players categorize acceleration together with rotation and translation. Thus, it would be interesting to see how to implement acceleration control through the widget — or if acceleration can actually be separated from the widget, through for example a a GUI button.

3.2.4 Clutching

[HPGK94] mentions that most spatial interfaces incorporate some type of*clutching* mechanism, e.g., a button that tells the system when to track the input device. The car-shaped widget from *AppMATes* consists of two clutches: *(1)* whether it touches the touchscreen and *(2)* whether the user is touching the conductive parts of the widget. Both need to be true for the widget to be tracked.

From our initial testing, the two clutches should result in different, separate feedback, according to the expectations of the participants. Participants understood that when they didn't touch the widget, the car wouldn't drive, but thought it was still being tracked. For potential solutions, it would therefore be relevant to investigate how to minimize these clutching issues.

3.3 Travel in Virtual Environments

As the main problems we found with *AppMATes* are related to travel, it is relevant to investigate travel in virtual environments. Much research in this field is specific to applications with six degrees of freedom, such as in virtual reality. Therefore, we only focus on the more general research that we believe fit our specific context.

[Bla11] state that one of the most common and fundamental interaction tasks in 3D interfaces is travel. They define travel in the virtual world as actions that allow the user to control the position and orientation of the viewpoint through, for instance, translation, rotation, and velocity control. They also define travel as the *motor component* of navigation, while *wayfinding* is the *cognitive component* of navigation, which focuses more on the planning and decision-making about where to go. Since the problems we found in *AppMATes* are primarily concerned with positioning the viewpoint, this project focuses on travel and not wayfinding.

[BKLJP04] state that travel is often used in combination with *primary tasks*, such as picking up treasures, fighting enemies, or obtaining critical information. Thus, users should not need to think about how to turn left or move forward, as this might distract them from their primary tasks. The authors state that this increases the need for usability in travel techniques.

[BKLJP04] present three different types of travel tasks: *exploration*, *search* and *maneuvering*. In *exploration tasks*, the user has no explicit goal, and the main purpose of travelling is to gain information about the space, including the objects and locations that exist within it. Exploration is therefore typically done in the beginning of an interaction with an environment. The authors state that interaction techniques that support exploration should allow continuous and direct control of the viewpoint movement. Furthermore, interaction techniques for this task should have little cognitive load, so the user can focus on learning about the world. *Search tasks* are similar to exploration tasks with the difference that the user has a specific goal in the form of a target location. Here, people can either make use of *naive search*, where they are unaware of the position of the target or the path to it, or *primed search*, where they have

knowledge of the target position beforehand. *Maneuvering* is a subtask of travel that takes place in a local area where precise movements are essential. Here, it is important that the viewpoint can be positioned precisely according to the task that needs to be performed. If not, this might result in users failing to perform the task and becoming frustrated.

In *AppMATes*, all three types of travel tasks are primarily used as travel tasks. When players start the game, they can explore by driving around in the world. They can also get search-based tasks, where they are asked to find certain objects in the world, using an arrow that indicates the destination. Players can also collect points represented by wheel caps. Children had a tendency to search for these as soon as they found out that they existed (naive search). Furthermore, players could enter a race against other cars, where the focus was on maneuvering. However, most of the time was spent with exploration and naive search tasks. Therefore, these tasks are also the focus for potential solutions.

To facilitate the completion of the above-mentioned travel tasks, we need travel and interaction techniques that fit these tasks. Since the context is videogames, it is relevant to investigate how travel interaction techniques are used in other games. As mentioned before, travel concerns the control of the virtual viewpoint. The following section investigates how virtual cameras are used in games.

3.4 Cameras in Games

As videogames become more complex, they tend to include bigger and bigger worlds. Unlike arcade games that typically display just a single screen, games today have the possibility to show virtual worlds that are bigger than what can be displayed on one physical screen. Depending on the game genre, this can be achieved in various ways.

[Ker15] offers an overview of camera systems used throughout the history of digital *scrolling games* (commonly referred to as *2D* or *2.5D* games). Since there is a lack of a vocabulary within the field of videogame cameras, he decided to make his own terminology in order to create a foundation that can be used by game designers. More specifically, he describes *scrolling* or *panning* cameras. These refer to any attempt to display a virtual scene that is larger than what fits on the screen in which players can't move in the z-axis.

Depending on how the game is designed, the typical goal is to show the player what is located

further ahead. By using a scrolling camera, the game designer has control over what the camera shows and how it should move. A common solution is to have the camera lock on to the player (referred to as *position-locking*). This means that the player is always in a fixed position on the screen. In games with *position-locking*, the players' screen position is often locked to the side of the screen opposite the direction they are travelling (referred to as *static-forward-focus*). This way, players can see what is coming ahead and ignore what is behind them. In games where the player can travel both left and right, *position-locking* is often implemented using *dual-forward-focus*, in which the camera has two separate lock-on positions, depending on the orientation of the player.

In some cases, the designer might want to give players the possibility of moving their avatar without moving the camera. Lacking any *forward-focusing*, this is simply an invisible bounding box on the screen, referred to as *camera window*. As long as the players move inside the bounding box, the camera doesn't move. When crossing the boundaries, the camera goes into *position-locking*.

When the camera moves, it needs to move with a certain speed. With *position-locking*, if the camera speed is instantaneous, the camera is always completely fixed on the player. If paired with, for example, *dual-forward-focus*, the camera would jump between the camera positions. Here, it would make more sense to either move the camera to the other position at a fixed speed, or, for example, make the transition dependent on the character's movement in that direction. Likewise, *lerp-smoothing* could be implemented where the speed of the camera decreases the closer it gets to its destination. For an alternative smooth transition, it's worth looking at *physics-smoothing* where the *lerping* is also dependent on the camera's current velocity, resulting in a camera that both accelerates and decelerates. *Lerp-smoothing* and *physics-smoothing* both result in good-looking, smooth motion of the camera, but are thereby less responsive and not necessarily good choices in highly intensive games.

[Ker15] also pinpoints three main challenges when working with scrolling cameras:

- Attention: what the player needs to see (driven by the game designer)
- Interaction: what the player wants to see (driven by the player)
- Comfort: how to reconcile these needs smoothly and comfortable (driven by the systems reacting to the player)

In relation to *AppMATes*, the main problem is that players often are unable see what is in front of and next to them. Furthermore, there were cases where players weren't able to see the road they were driving on due to the camera being placed poorly. Therefore, in many cases players cannot see what they want and need to see, and they can't easily make it better, which means the problem is related to all three challenges.

3.5 Summary

Throughout the different sections in this chapter, we have found knowledge that can support the design of our future solutions.

The target group for this project has been chosen to be children between the ages 5 and 7. Since *AppMATes* is based on the feeling of controlling a toy car, where there is a strong link between the physical and virtual, the same relationship between the physical and virtual should be present in our solution. In other words, the *aesthetics* of feeling like controlling a car should persist [HLZ04]. To encourage this as much as possible, we will use the same tangible widget (the *Disney AppMATes* car) in our solution. Furthermore, our solution should be tested in a free-roaming environment that imitates the one used in *AppMATes* with primary travel tasks that focus on exploration and search. The main purpose of this solution is to solve the problem of players not being able to see what is located in front of and next to the widget due to it being too close to screen edges. Other problems described in this chapter, such as clutching and maneuvering the widget, will be investigated as well. Finally, this solution needs to support the feeling of controlling a car. This means that the interaction technique must make sense in the context of a car.

Chapter 4

Design and Implementation

We chose to build our application from scratch in the Unity game engine. Besides designing and implementing new camera techniques, we had to implement a basic system to read touch points and identify the widget. We will describe our implementation in the following. Furthermore, we also describe each of our interaction techniques. These were designed based on knowledge gained from Chapter 3, mainly by taking inspiration from the initial observations of children playing *AppMATes*. The goal of our tests was to find the best solution to counter the problems we found with the *AppMATes* interaction technique. In the end, we will pick the best of our designs and conduct a more thorough test with it against *AppMATes*. Throughout the project, we used the 2015 model of the 12.9" iPad Pro, which has a resolution of 2732x2048 px (264 DPI). Figure 4.1 shows the *AppMATes* widget on the iPad Pro.

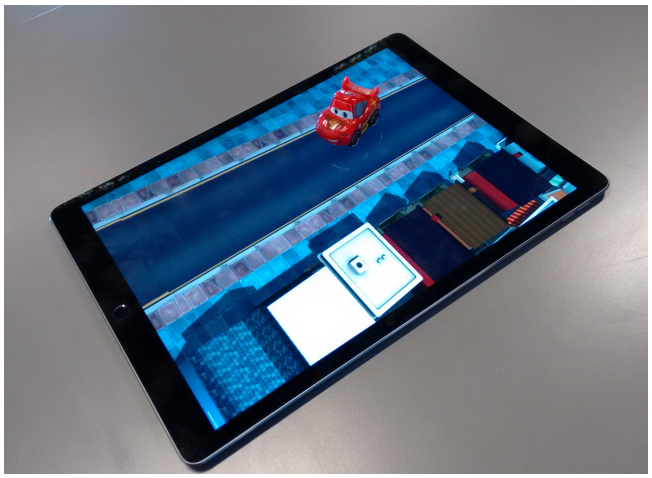


Figure 4.1: We used the 2015 12.9" iPad Pro for this project.

4.1 Widget Implementation

The following chapter explains how we detected the widget by implementing an algorithm using C# in Unity.

4.1.1 Disney AppMATes

On the underside of the *Disney AppMATes* widget are three detection points (see Figure 4.2). Touching or holding these conductive surfaces and placing the widget on a touchscreen enables the iPad to register the detection points of the widget as touch points. From these, we calculated a screen position and orientation of the widget. The detection points on the *AppMATes* widgets form a unique isosceles triangle for each *AppMATes* widget in the *AppMATes* series. We call these points the *front-*, *leftleg-* and *rightleg-point*.

Figure 4.2: The underside of the *AppMATes* widget has three detection points (indicated with green circles) as well as three lenses that are used to capture light from the screen and project it out to simulate car headlights.

4.1.2 Measurements

To be able to detect the widget, we first needed to know its measurements. The algorithm we developed needed the distances and angles between each detection point. Since the iPad is not able to detect these measurements precisely, we needed to take into account how much the tablet measurements would potentially deviate from the widget's actual measurements.

Using Unity's touch points, we measured the widget using the tablet and noted down the minimums and maximums for the lengths between the detection points (seen in Table 4.1). These lengths were used to indicate an acceptable spectrum for identifying the widget. Furthermore,

	Shortest	Longest	Difference	Mean
Front-to-Leg Length	1.96	2.28	0.32	2.12
Leg-to-Leg Length	1.09	1.37	0.28	1.23

Table 4.1: Lengths in centimeters between detection points on the widget measured using the tablet. *Shortest* and *longest* show the extremes.

we measured the angle differences between the two leg angles and found the biggest difference they could have was 13. We noted that as the angle confidence threshold.

4.1.3 Detection

To detect movement and orientation of the widget, the algorithm needs to detect the widget in each update frame. Furthermore, because of the design of the *AppMATes* widget, the user can accidentally tip the widget, thus losing connection to one or more of the detection points. Therefore, the algorithm also needs to take into account scenarios in which the tablet can only detect two of the detection points. The detection of the widget is separated into three phases:

1. Calculate a temporary point, if only two touch points can be seen.
2. Find all isosceles triangles.
3. Find the isosceles triangle that sufficiently matches the widget.

If the iPad registers more than two touch inputs, the algorithm tries to determine whether any set of three touch points could be the widget. If there's only two touch input, it checks for the previous known triangle, if any.

To determine whether a triangle is an isosceles triangle, we check if two of the triangle's three angles are approximately the same, by comparing their absolute differences against the confidence threshold of 13 degrees (see Section 4.1.2). For every pair of approximately-identical angles, the corresponding triangle is copied to a list, and the two points flagged as the legs of the triangle. This means it's possible for up to three isosceles triangles to be copied from one set of three touch points. If no isosceles triangles are found, the algorithm ends and no widget is detected.

For each triangle found this way, we determine which ones look sufficiently like the triangle of the widget. The distance between each pair of points is calculated and converted to centimeters,

using the iPad's DPI. If all the distances of the triangle fit inside the spectra (as described in Table 4.1), the triangle is saved. If more than one triangle is saved this way, the triangle with the smallest sum of absolute differences between its distances and the mean distances (as described in Table 4.1) is chosen. If no triangle was saved doing this step, the algorithm ends and no widget is detected.

From this triangle we calculate a screen position and an orientation. The screen position of the widget is simply the mean position of the three touch inputs that constitute the triangle. This point approximately corresponds to the center of the widget figurine. The orientation is the direction vector from the widgets screen position to the frontpoint (see Figure 4.3). These screen positions and orientations are what will be used to calculate the virtual representation of the widget in the game. Furthermore, we flag which leg is the left and which is the right. This information is needed in case the iPad loses connection to one of the touch inputs associated with the widget.

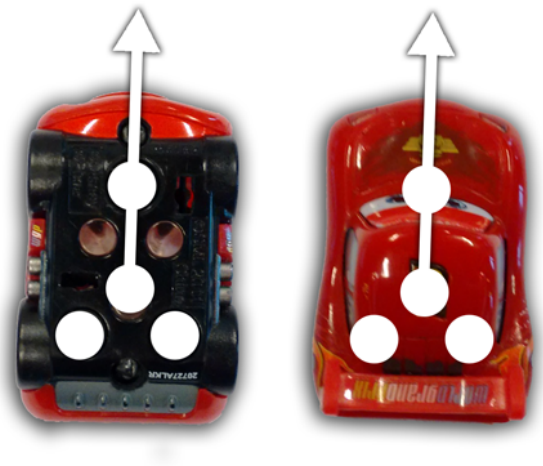


Figure 4.3: Calculated position and orientation of the widget. Center point indicates the position.

In the beginning of each update frame, if there are only two touch input registered, and we found the widget in the previous frame, we check if the two points are considered the same as two touch inputs previously associated with the widget.

Through the Unity API, each touch input is assigned a unique touch ID when registered. Across frames, this means it's possible to keep track of which touch inputs are new and which have recently moved. Using these IDs, we can see whether the iPad has lost connection to a touch point associated with the widget.

From only two touch points, it's possible to create a temporary third point to calculate a full triangle. Because there are two possible positions for this calculated point, using the flags of the two known touch points it's possible to determine which of the two possible positions is the correct one.

Unfortunately, the touch points sometimes get mixed up and seem to switch places. This is most likely due to the close proximity of the touch points. This means tipping the widget or pushing it slightly over the edge of the screen in some cases creates a wrong representation of the widget. This results in the car being positioned and oriented incorrectly, as long as there's only two touch inputs. We determined this was preferable to completely losing connection to the widget.

4.2 Interaction Techniques

Throughout the project, interaction techniques have been tested iteratively with qualitative tests conducted in a similar fashion as the initial *AppMATes* test (see Chapter 2). 22 children with an average age of 6.6 years ($SD = 1.9$) participated in these iterative tests. The children were recruited at a local library. Furthermore, a few adults have been used for some smaller tests for minor tweakings. A between-subjects design was used to avoid bias, as we experienced that children were heavily influenced by the first interaction technique they experienced. Therefore, each child tried only one interaction technique.

In the following subsections, we explain all of our designs. Each design is backed by an explanatory illustration. Figure 4.4 shows a legend of the icons we use in these illustrations.

Figure 4.4: The following illustrations will make use of these icons.

4.2.1 AppMATes (Our Implementation)

When we mention the *AppMATes* solution, we are referring to our implementation of the camera controls found in the free-roaming sections of *Disney AppMATes*. As the project is not a modification of the game, we had to imitate Disney's solution as accurately as possible. In *AppMATes*, the camera does not rotate, and its position is independent of the widgets screen position. Instead, the virtual widget accelerates and moves the camera in the direction of the orientation of the widget (see Figures 4.5 and 4.6). This control scheme was used as the basis for every other proposed solution. Assume that all upcoming solutions build upon this foundation unless stated otherwise.

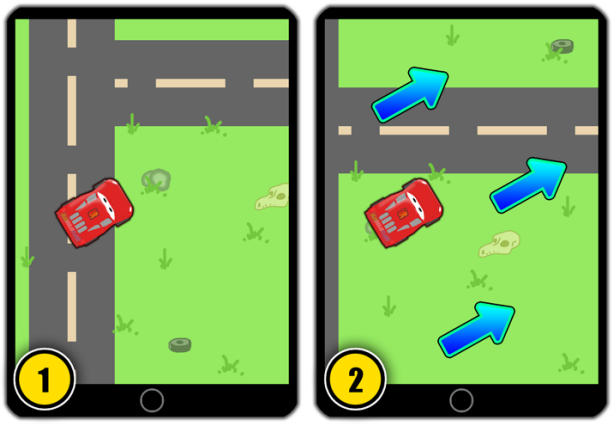


Figure 4.5: *AppMATes*: *(1)* The game looks at the orientation of the widget, and *(2)* the camera then accelerates in this direction.

When working on the *AppMATes* imitation, certain design restrictions became apparent. The go-to solution of *position-locking*, as described in Section 3.4, is not directly applicable. As the player can freely move the widget anywhere on the screen, the camera can no longer force the

```
float Acceleration = 10;
float MaxSpeed = 15;
float Speed = 0;
. . .
Speed += Acceleration*Time.deltaTime;
Speed = Mathf.Min(Speed, MaxSpeed);
Vector3 Velocity = carTransform.forward * Speed;
cameraTransform.position += Velocity * deltaTime;
```

Figure 4.6: *AppMATes* calculation. *Acceleration* and *MaxSpeed* were tweaked until they felt like the original *AppMates* game.

player character to stay in a certain area of the screen. Likewise, applying any smoothing on the camera movement influences the velocity of the player character. The *AppMATes* solution avoids these complications by moving the camera according to the speed of the player character and the widget's orientation. By doing this, the widget's position on the screen has no effect on the camera movement.

4.2.2 Ground Dragging

This interaction technique comes close to the *AppMATes* solution, with the exception that the camera can be further manipulated by dragging the widget. Dragging the widget on the screen moves the camera in the opposite direction at the same speed, effectively making the car stay in place in the virtual world. This can be used for "dragging the ground", like one would drag the map in, for example, Google Maps. Likewise, lifting and repositioning the widget avoids this. This means players can use the widget to orient themselves. The idea is that by dragging the widget, it is possible to move the camera around and explore the environment more (see Figure 4.7). This interaction technique was also partly inspired by [BKLJP04], who proposed a technique called *Grabbing the Air*, in which the user grabs and drags the viewpoint to a new location.

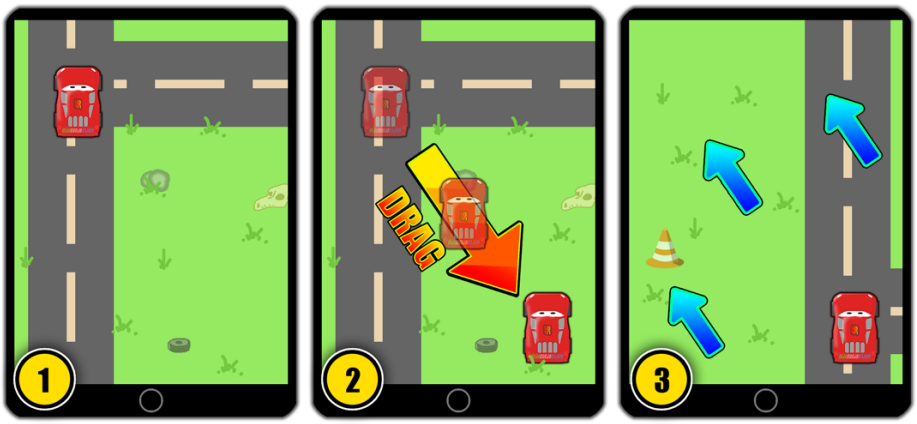


Figure 4.7: *Ground Dragging*: *(1)* If the widget is not moved, it acts like *AppMATes*. *(2)* By dragging the physical widget, *(3)* the camera is moved in the opposite direction at the same speed. This keeps the virtual car in place in the virtual world.

Evaluation of ground dragging

Most of the children had a hard time understanding how to move around. They didn't seem to understand the dragging metaphor. Because of this, the children tended to place the car

at the bottom of the screen without moving it around much. When moving the widget, some children mentioned that it caused the game to look "weird".

4.2.3 Flip-View

This technique was inspired by older games where it was technically challenging to have a constantly-moving camera. The solution was to use a static screen that only changed if the player character moved to the edges of the screen. If they did, everything would pause for a brief moment while the camera either jumped or slowly panned to the side to reveal a new location (or screen), hence the original term *flip-screen*. In essence, the world is divided into a grid, which the camera flips between. An example of this is the original *The Legend of Zelda* for the Nintendo Entertainment System. In *Flip-view*, when the player moves the widget across predetermined boundaries close to the edge of the screen, the camera scrolls in that direction for a fixed distance (see Figure 4.8). As long as the widget stays inside the boundaries, the camera does not move; hence, the idea is that players can decide if they want to interact with the objects currently shown on the screen, or if they want to move to a different location. A related concept was proposed by [Bar00], who describes an interaction technique called *Rock 'n Scroll*. It uses a button to avoid unintentional scrolling, meaning that it has a clutching mechanism. In *flip-view*, the act of moving the widget closer to the screen edge becomes a clutching mechanism.

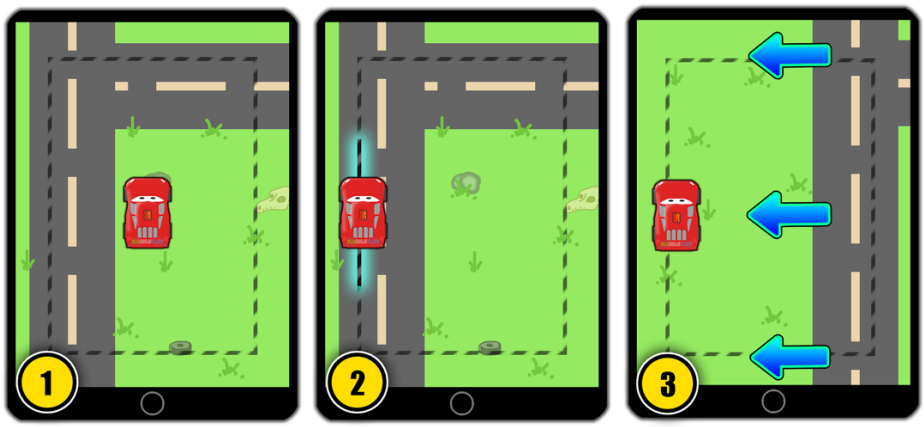


Figure 4.8: *Flip-View*: *(1)* When the widget is placed inside the boundaries, nothing happens. *(2)* If the widget is pushed or placed outside the boundaries, *(3)* the camera scrolls in the corresponding direction for a set amount of time.

Evaluation of Flip-View

Compared to the other interaction techniques, the *flip-view* technique seemed to be slower and less appealing for the children. In the beginning, it was quite confusing for them to understand that they needed to place the widget close to the screen edges, but after a couple of tries they seemed to grasp how it worked. That being said, it was clear that this method seemed tedious and unrealistic. Having the car metaphor in mind, this way of moving made little sense to the children. Furthermore, as opposed to the original usage of it in classical arcade games, because of the physical widget still being at the edge of the screen after the scrolling, the virtual widget would effectively have teleported into the new location. This kind of interaction would have worked better if there was enough to do on a single screen, and if the widget resembled an omnidirectional vehicle. However, one thing we noticed was that due to the interpolated movement being a bit slow, children would often impatiently try to drag the widget forward as if it made the car move faster. This dragging motion revealed a possibly more natural and intuitive way of interacting with the widget, which inspired the *acceleration dragging* interaction technique described further below.

4.2.4 Speed Relative to Screen Center

This interaction technique is based on the relative distance from the center of the screen, meaning that the further away the widget is positioned from the center, the faster the car drives, but still in the direction of the widget's forward direction (see Figure 4.9). This concept is closely related to an analogue stick on a game controller. Furthermore, this takes inspiration from Section 3.4 and the *lerp-smoothed position-locking*, in which the camera is trying to catch up to the player character. Unlike the other techniques, here it is possible to regulate the virtual widget's speed.

We implemented two versions of this: in the first version, the car stopped if the angle between the forward direction of the widget and the direction vector from the center of the screen to the widget exceeded 90 degrees. In the second version, the dot product of these two vectors, clamped to [0-1], was used as an extra scalar on the speed. In both cases, this meant that if the player placed the widget next to the center of the screen and pointed the widget towards the center, the car wouldn't move. The intention of this technique was to give the player more

control over the speed of the car.

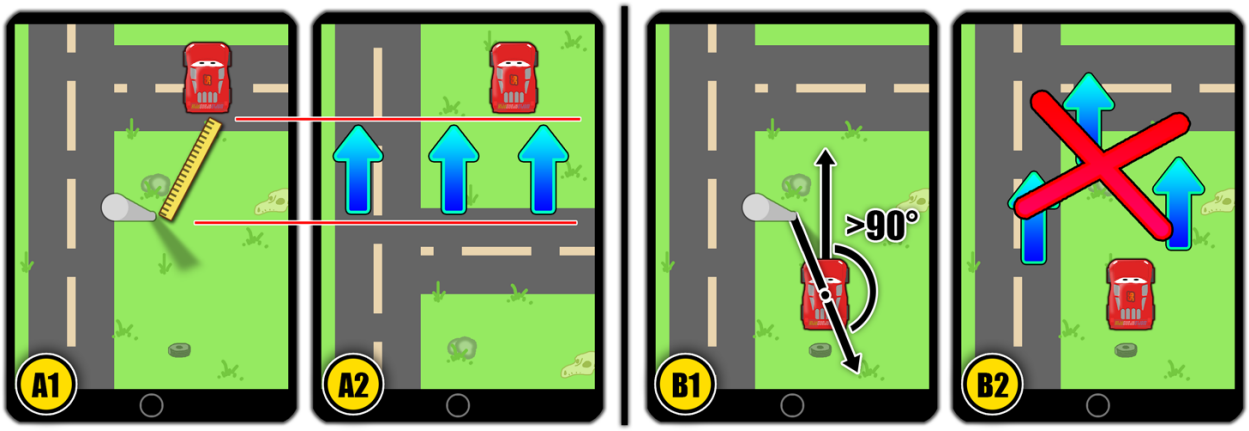


Figure 4.9: *Speed Relative to Screen Center*: *(A1)* The game looks at the distance from the center of the screen to the widget, *(A2)* and then moves the widget with the corresponding speed in the direction of the widget's orientation. *(B1)* If the angle between the forward direction of the widget and the vector from the screen center to the widget is above 90 degrees, *(B2)* the virtual car does not move.

Evaluation of Speed Relative to Screen Center

Unfortunately, it motivated test participants to position the widget close to the screen edges in the direction that they wanted to drive. E.g., to drive upwards quickly, one would position the widget in the top of the screen. This downside was greater than partially solving the problem of not being able to control the speed of the virtual widget. However, this gave inspiration to a reverse version in which the car stops if the widget pointed away from the center. This can be seen in the solution *reverse speed relative to screen position*.

4.2.5 Reverse Speed Relative to Screen Center

As with the standard version, speed is controlled by the widget's proximity to the center of the screen. But instead of stopping when pointing towards the center, the car stops when pointing away from it. This means that if players want to go upwards, in addition to pointing the widget upwards, they have to pull the widget downwards. The effect looks a little like *projected-focus* described in Section 3.4, but in an inverse form; the player character moves forward because the camera (relative to the player character) moves forward. The intention is that this lets players naturally give themselves more visible space in front of them.

Evaluation of Reverse Speed Relative to Screen Center

This reverse version only partly succeeded. When children seemed to be conscious about how it worked, they could control the car and stayed away from the edge in the direction the widget was facing. But as soon as they wanted to drive somewhere visible on the screen, or follow the road, they forgot about it, moved the widget to where they wanted to go. They were confused why the car stopped, even though they previously seemed to understand the concept. From this we started seeing a pattern. Children (and adults) wanted to drive on roads and be able to instantly move the widget to somewhere visible on the screen. This desire seemed to overwrite any understanding they had of the control scheme introduced to them, and they intuitively reverted to how *AppMATes* is controlled, even though they had never tried that solution and it didn't work with the current control scheme.

4.2.6 Acceleration Button

With the *AppMATes* solution, the majority of participants didn't intuitively understand that the two clutches (holding the conductive surfaces and placing the widget on the screen) didn't control separate parts. They thought that placing the widget on the screen meant the iPad could track it, and that touching the conductive surface of the widget made it drive. This could be seen when asking the participant to stop the car, to which they just let go of the widget. We had seen with the original *AppMATes* game: that children could easily use the horn while driving, by pressing a GUI button located in the lower left corner of the screen. In an attempt to clearly separate the detection of the widget and the acceleration of the car, we implemented an acceleration button at the bottom of the screen (see Figure 4.10). Holding down this button made the car accelerate. The assumption was that this meant the two clutches would be connected with detection of the widget only in a more obvious way, and players could easily control the acceleration of the car.

Evaluation of Acceleration Button

Holding down a button to accelerate, separate from the widget, seemed to bring with it much more cognitive load than anticipated. Children constantly forgot the button and would instinctively lift the widget and let go of the acceleration button in high-pressure situations, instead of just letting go of the acceleration button. When putting down the car on the screen, they

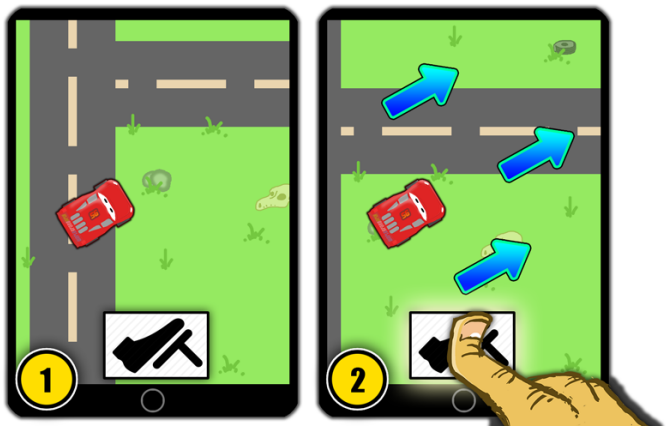


Figure 4.10: *Acceleration Button*: *(1)* Placing the widget on the screen no longer makes it drive. *(2)* Pressing the accelerate button makes the car accelerate and behave like *AppMATes*.

temporarily forgot about the acceleration button and got confused why the car wasn't moving, despite not having tried the *AppMATes* solution. From this we could conclude that we had to be aware of multitasking.

4.2.7 Move-Rotation

A common problem we found with *AppMATes* was the constant need to turn the widget so much that participants either grabbed the widget awkwardly, the iPad lost connection to the widget, or participants had to temporarily let go of the widget to reposition their grasps. A possible way to combat this was a way to rotate the camera, so they didn't have to rotate the widget as much.

Taking note from the problems with the *acceleration button*, the control of the camera rotation needed to be implemented through the widget. We chose to control the rotation through the horizontal alignment of the widget. The more horizontally-aligned the widget is, the more it will rotate in the direction it is pointing (see Figure 4.11). Hence, if the widget points slightly to the right, the camera rotates slightly clockwise.

Evaluation of Move-Rotation

Move-rotation had many of the same problems as *AppMATes* and only seemed to positively impact the amount of times players had to excessively turn the widget. On the negative side, players seemed to have a harder time orienting themselves and drove more aimlessly. However, when driving on a curved road, these problems did not occur, and instead players seemed to

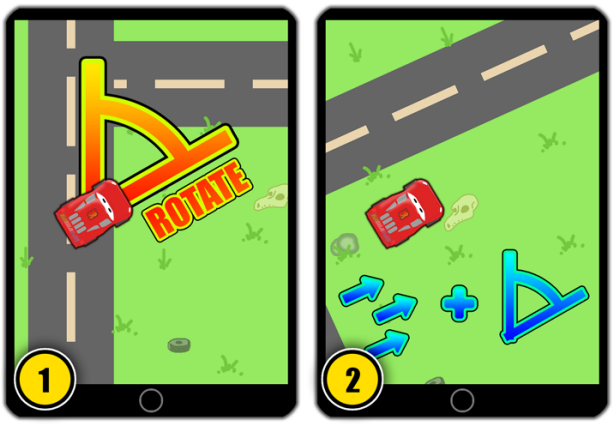


Figure 4.11: *Move-Rotation*: *(1)* Rotating the widget, *(2)* makes the camera rotate accordingly, as long as the widget is not vertically aligned.

have the same difficulties with staying and orienting the widget on the road as with *AppMATes*. They had less difficulties holding the widget because of the decreased need for turning it as much. This meant the iPad lost connection to the widget less often. In the original *AppMATes* game, the creators ostensibly also saw this, as it seems their solution, when driving on a race track, was a mix between turning the camera to face the same way as the car and having the camera on rails, both in terms of position and orientation.

4.2.8 Forward Edge Proximity

Taking inspiration from previous attempts with *reverse speed relative to screen center*, we still wanted to discourage players from driving too close to the edge of the screen in the direction the widget was facing. *Forward edge proximity* slows down the car when the widget approaches the edge in its forward direction (see Figure 4.12). The intention with this approach is that it's slightly more forgiving than the *reverse speed relative to screen center* solution, since the area where the car slows down is smaller. Therefore, *forward edge proximity* might be easier to adapt to.

Evaluation of Forward Edge Proximity

The results were similar to those with *reverse speed relative to screen center*. When something of interest appeared in participants' field of view, or if they wanted to follow a road, they would forget about the limitations of the solution, resulting in frustrating scenarios where the car stopped or slowed down.

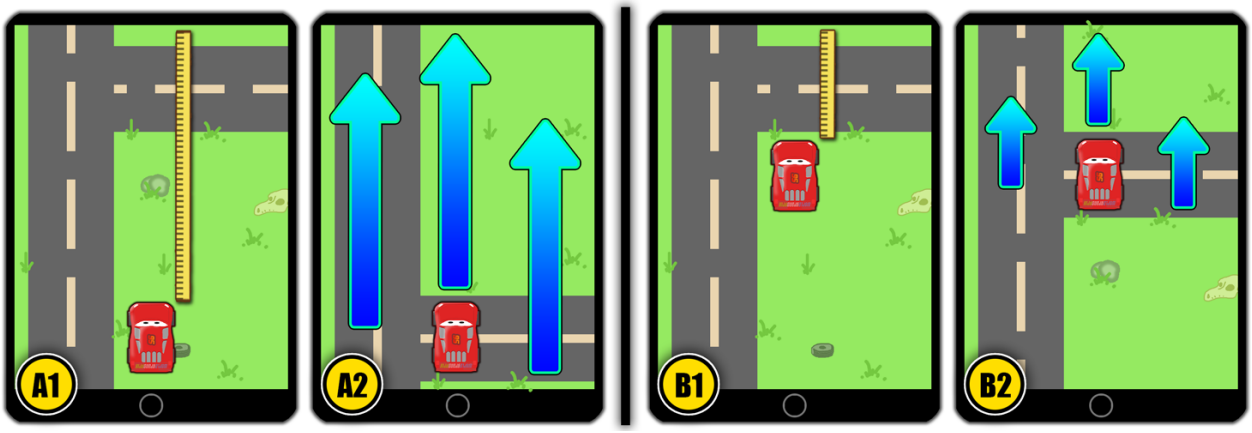


Figure 4.12: *Forward Edge Proximity*: *(A1)* If the car is placed far away from the forward edge, *(A2)* it has a high max speed. *(B1)* Placing or pushing the widget closer to the forward edge *(B2)* reduces the max speed of the car accordingly. Pushing it the closest possible to the forward edge reduces the max speed to 0, making the car stop.

4.2.9 Acceleration Dragging

Inspired by children's' apparent predisposition to begin dragging frantically when the car wasn't accelerating quickly enough in the direction they wanted to go (as well as the results gathered from the *flip-view* solution), we let players control the acceleration of the car through dragging (see Figure 4.13). Imagine a pull-back toy car where you have to pull back or push forward the car to make it speed off — except the acceleration would happen while dragging. We implemented this in two ways: one in which you pull back the car and one in which you push it forward. The intention with *acceleration dragging* was to give players a way to control the speed of the car and see how this control scheme would affect their positioning of the widget on the screen. We hoped players would quickly push or pull the widget and then lift and reposition it where they wanted to drive from, preferably close to the center of the screen.

Evaluation of Acceleration Dragging

Pushing the widget forward to accelerate resulted in players slowly pushing the widget forward (as opposed to the expected push, lift and repositioning) until it slowly hit the edge of the screen and stopped. Many children had a hard time understanding the pull-back analogy when pushing the car. However, the pull-back version had better results where players were forced to move the widget backwards to move forwards (inverse). One child even mentioned that it felt a bit like using a skateboard where you had to "kick" back a couple of times to get speed

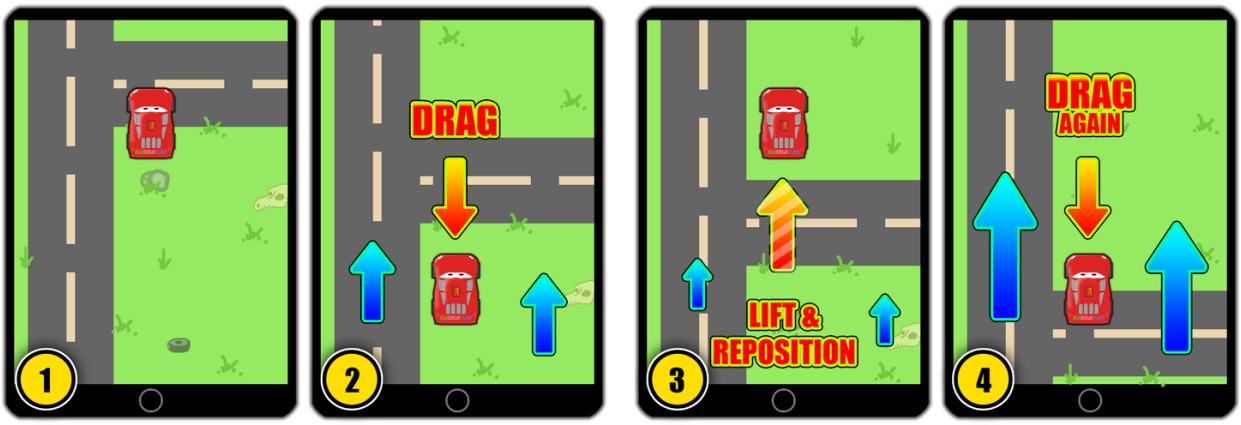


Figure 4.13: *Acceleration Dragging*: It works a bit like a pull-back toy. *(1)* Placing it on the tablet does not make it move. *(2)* Dragging it backwards makes the camera accelerate forward. *(3)* Lifting and repositioning let's you move the car without affecting the speed. *(4)* Dragging the widget back again applies more acceleration, until it hits max speed.

and then let the car drive for a few seconds before "kicking" it in motion again. That said, as soon as they stopped dragging and let the car drive, they would still push it forward to the edge they wanted to go. When they turned the car, they would often momentarily forget that they needed to pull the widget back to accelerate and not forward, even though it was the only version they had tried. Furthermore, children seemed to get exhausted with this solution.

4.2.10 Road Focus

A reoccurring problem through many of the proposed solutions was that children and adults alike seemed to have a predisposition to want to always drive on the roads. Driving on the roads often meant that the widget would end up in the corners of the screen or with its side too close to the edge of the screen. This meant that participants often couldn't see a road that was turning off-screen, or that they couldn't even turn down that road. When driving down the side of the screen on a road, players would rarely align the car properly with the road. This often resulted in the road slowly going off screen and thereby forcing the player off the road to their frustration. We chose to try and exploit this apparent predisposition.

With *road focus*, if the car is driving on a road, the camera slowly drifts towards the car, depending on how far away from the center of the screen the widget is (faster the further away it is). The result is that if they drive on a road close to the side edge, the camera will slowly bring the road closer to the center of the screen (see Figure 4.14). To avoid affecting the speed of the car, the drifting is only applied along the horizontal axis of the virtual car. This will

look like the car is drifting to the side a bit, but presumably so little that players won't notice. The assumption is that this solution will help keep the widget away from the edge of the screen — especially the corners.

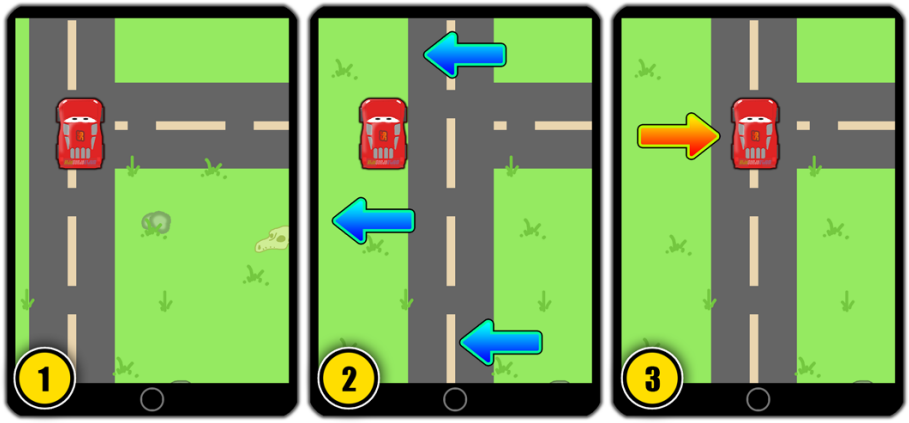


Figure 4.14: *Road Focus*: *(1)* When driving on a road alongside the side edge, *(2)* the camera slowly drifts towards the virtual car, pulling the road towards the center. *(3)* The assumption is that players themselves will reposition the widget on the road.

Evaluation of Road Focus

As intended, the solution brought the roads closer to the center of the screen. As we expected, players would follow the road, and, in doing so, would not drive as close to the edges of the screen as before. In opposition to earlier tests, participants tried the *AppMATes* solution first, followed by *road focus*, to see how noticeable the solution is, since it is so close to the *AppMATes* interaction technique. During *road focus*, when asked if they could feel any differences between the two controls schemes, participants replied "no", insinuating the help was invisible. We could not definitively say at this point that *road focus* actually helped significantly in these

```
float MaxSpeed = 15;
float PushAmount = 1.5f;
Vector3 newDir = widgetViewportPosition - viewportCenter;
newDir = cameraTransform.TransformDirection(newDir);
newDir = Vector3.ProjectOnPlane(newDir, Vector3.up);
newDir -= Vector3.Project(newDir, carTransform.forward);
newDir *= MaxSpeed * deltaTime * PushAmount;
cameraTransform.position += newDir;
```

Figure 4.15: *Road focus* calculation. *PushAmount* was tweaked until it felt right at *1.5*. *MaxSpeed* was set while making the *AppMATes* solution.

scenarios, but nonetheless it warranted further investigation.

4.2.11 Conclusion

The purpose of this project was to find a solution to, or mitigate problems with, the *Disney AppMATes* control scheme and its camera controls. These problems ranged from players pushing the widget too close to the edge of the screen, to misinterpretations of the clutching of the widget, to general problems holding and maneuvering the widget. Through an iterative design process, we've implemented and tested nine different proposed solutions.

Through these solutions we have not been able to surpass the intuitiveness of the *AppMATes* solution. In all cases, except *road focus*, *move-rotation* and *acceleration dragging*, children were more naturally-inclined to engage with the game as if it was the *AppMATes* solution, than the actual implemented solution, even though they had never tried *AppMATes*.

To combat the lack of acceleration and speed control, we tried (*reverse*) *speed relative to screen center*, *acceleration button* and *acceleration dragging*. The multitasking expected with *acceleration button* seemed to be too much for the children. From this we could conclude that any future solutions we implement had to revolve purely around the widget. When children used *speed relative to screen center*, they had a hard time grasping the concept. But even that small bit of understanding was cast aside when they wanted to interact with objects on the screen or follow the road. *Acceleration dragging*, as one participant noted, felt like "kicking" a skateboard into motion. This would give them some control of the acceleration, but every participant consistently accelerated to maximum speed. We deemed none of these proposed solutions were good enough for further investigation into whether they could fix the problem of a lack of acceleration control.

Ground dragging, *reverse speed relative to screen center*, *forward edge proximity*, *acceleration dragging* and *road focus* each tried to tackle the problem with the players placing the widget too close to the screen edges. *Reverse speed relative to screen center* and *forward edge proximity* tried to discourage players by punishing them for driving too close to the edges. This only caused frustration for the children. *Ground dragging* tried to make movement of the widget not translate to movement in the game, in the hope it would discourage players from trying to make the car go faster by pushing the widget forward; instead, it confused the participants.

Acceleration dragging introduced a way to control the speed of the car, but it also forced players to drag the widget backwards for the car to move forwards. Even though players understood this dragging mechanism, it was still not as intuitive as the *AppMATes* solution. Post dragging, the children still pushed the widget forward. With *road focus*, we stopped trying to solve this particular proximity problem and instead started looking at the widget's proximity to the edge of the screen it was driving alongside. With *road focus*, we seemed to successfully exploit players' desire to drive on roads and keep them from driving alongside the edge of the screen, while keeping the more intuitive *AppMATes* controls.

From these findings, we chose to further investigate *road focus* and test whether it made a significant change in the widget's proximity to the edges of the screen, and how else it might impact the experience.

4.3 Final Design

During the iterative design process, in which we tried to find a suitable solution for one or more of the discovered problems with *AppMATes*, we tried to tackle *(1)* the lack of acceleration control, *(2)* losing your grasp on the widget and *(3)* driving too close to the edges of the screen. From preliminary tests on each proposed solution, we arrived at the *road focus* solution.

Road focus was designed to try and mitigate the instances where players place the widget too close to the screen edges. More specifically, *road focus* tries to stop the player from driving alongside the edge of the screen, e.g., with the widget's side up against the edge.

For *road focus* to work, we needed roads for the game world. We created a city with a grid-based road network (see Figure 4.16). In order to imitate the exploration tasks in *AppMATes*, the city has been built as a free-roaming environment. Most blocks are filled with buildings with the intention of making them look non-traversable, while some blocks infer the possibility of traversal (like an opening in a cluster of trees). We didn't want to explicitly tell participants whether they are allowed to — or can — drive off the roads.

In order to imitate the game environment in *AppMATes* as much as possible, and at the same time give the children a motivation to play the game, it was important to make the environment interesting to explore. For this reason, we used 3D assets (e.g., buildings, cars, and trees) with a colorful and cartoony artstyle. To encourage exploration, many of the city blocks have different,

Figure 4.16: The city is structured in a grid-like network.

visually-distinguishable buildings, which ensure there is something new to look at when driving around.

In order to give players a search task (see Section 3.3), and thereby give them a purpose for driving around, we implemented an objective arrow similar to *AppMATes* (see Figure 4.17). The idea was to guide players to different collectable objects. However, we noticed that children had a tendency to completely ignore driving on the roads with this task. It seemed like the pointing arrow completely removed their desire to drive in a realistic way, but instead they just followed it blindly. Therefore, we decided to remove the arrow and place golden coins that can be collected. This caused the children to drive around more naturally, while still having something to search for.

Figure 4.17: Initially, the game included a collect mission with a pointing arrow.

To make the environment feel responsive, we implemented various feedback effects, such as sounds when collecting coins and particle effects for driving on different surfaces. We also made other cars honk when the player drove close to them. Based on iterative tests, it seemed

like these effects were entertaining for the children. Furthermore, to encourage a joyful mood, pleasant background music played during the game.

For the final design, we implemented four levels in the game (see Figure 4.18). The first of these was a simple tutorial level, created to ensure that we got the most comparable results. Here, test participants can get used to the controls. This level was a small enclosed area of grass with a coin in each of the four corners.

Figure 4.18: Four levels were designed for the game.

The city is the main level where we want to compare the *AppMATes* solution to the *Road Focus* solution. Additionally, we want to see if this affection for roads only applied for roads specifically, or if we could somehow simulate the concept of a road, i.e., with a trail of coins. We had already seen in our preliminary tests that it also applied to clearings in forests. In these preliminary tests, children reacted positively to in-game coins, even though they had no apparent purpose. So we created a level with a road-grid very much the same as in the city, but made up of 184 coins instead (see Figure 4.19). The visuals were changed to a desert setting to contrast the city, but with less apparent city blocks (to avoid making the space between the blocks look like "invisible roads"). As in the city, this level encourages exploration and search tasks by having visually-distinct areas as well as coins to gather. The assumption was that we would see the same results in the off road desert level as in the city level.

Although the focus of this project is to make a solution to *AppMATes* for a free-roaming environment with exploration and searching tasks, we also wanted to see if *Road Focus* had a measurable impact on a closed environment with maneuvering tasks (see Section 3.3). There-

Figure 4.19: Desert world with trails of coins.

fore, we implemented a simple race track with curved roads and added other cars to it (see Figure 4.20). To make the situation feel urgent, fast-paced music is playing, while an animated stopwatch icon is shown in the top of the screen to induce the feeling of pressure. We had previously seen with the *AppMATes* game that, especially in the race, players would push the widget too much forward and would have a hard time following the road. The assumption was that *Road Focus* would help players stay on the road.

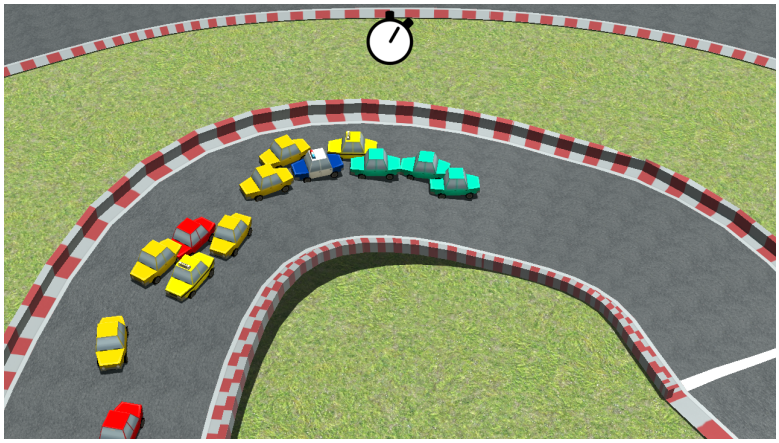


Figure 4.20: The race track provides an incentive to drive quickly.

Chapter 5

Experiment

This chapter describes how we tested and evaluated the *Road Focus* solution that we developed. We compare it against the default solution found in *Disney AppMATes*. We are mainly interested in reducing the distance from the widget to the edges of the iPad screen. Specifically, we have two types of edges: forward edges and side edges. These are illustrated in Figure 5.1.

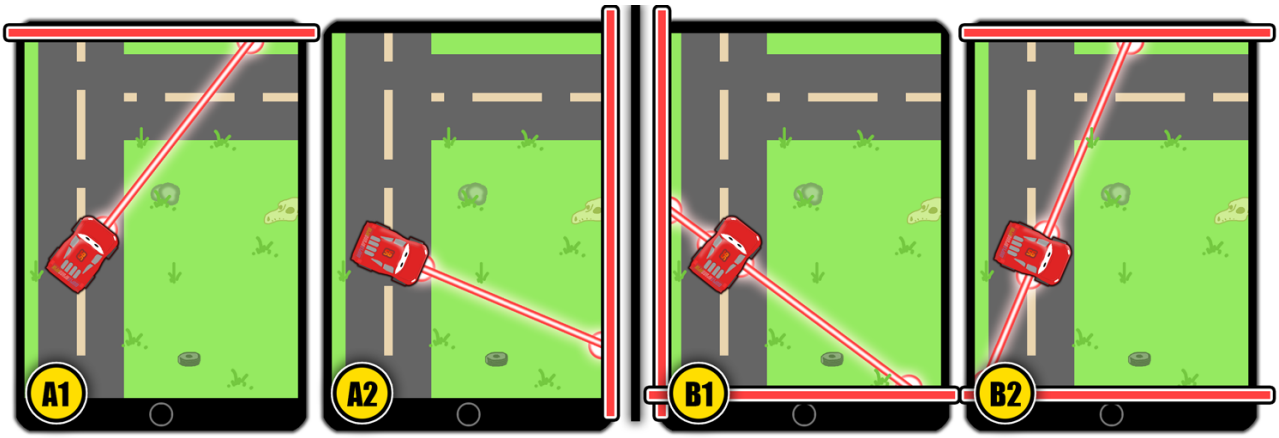


Figure 5.1: *(A1+A2)* Edge in the forward direction of the widget. *(B1+B2)* Edges in the side direction of the widget. Note that we only log the closest one of the two side edges.

5.1 Hypotheses

Based on the preliminary tests, we hypothesized that *Road Focus* will cause players to place the tangible widget further away from the iPad screen edges. The following are the hypotheses that we look into.

- HA_1 Test participants' average distance to the **forward edges** when using *Road Focus*

is greater than when using *AppMATes*.

- HA_2 Test participants' average distance to the **side edges** when using *Road Focus* is greater than when using *AppMATes*.
- HA_3 Test participants' frequency of getting 5 cm or closer to the **forward edges** when using *Road Focus* is smaller than when using *AppMATes*.
- HA_4 Test participants' frequency of getting 5 cm or closer to the **side edges** when using *Road Focus* is smaller than when using *AppMATes*.
- HA_5 Test participants drive more on roads when using *Road Focus* than with *AppMATes*.
- HA_6 Test participants have fewer occurrences where they drive off-center on a road while being 5 cm or closer to edges of the screen when using *Road Focus* than when using *AppMATes*.

5.2 Procedure & Levels

In order to compare our *Road Focus* solution with *AppMATes*, we implemented a game that, to our best effort, resembles the interactions found in *AppMATes*. Each of the levels that were created lasted for a specific time duration (except for the tutorial, where participants had to collect four coins to continue). The city and off road levels lasted 120 seconds each, while the race level lasted 90 seconds due to its simplicity. When the time ran out, the game would proceed to the next level. Data is logged for all levels except for the tutorial. The order of the levels was randomized. Each participant was randomly assigned an ID number. Based on this number, participants played with either *Road Focus* or *AppMATes*.

5.3 Test Participants

In order to recruit test participants, we made contact with four Danish schools and one kindergarten. We created a colorful poster to provide information about the experiment (see Figure 5.3). In total, we were able to recruit 64 children (37 boys and 27 girls), with an average age of 6.1 years ($SD = 0.5$). Children and their parents were notified beforehand and had to sign a consent form to participate in the experiment.

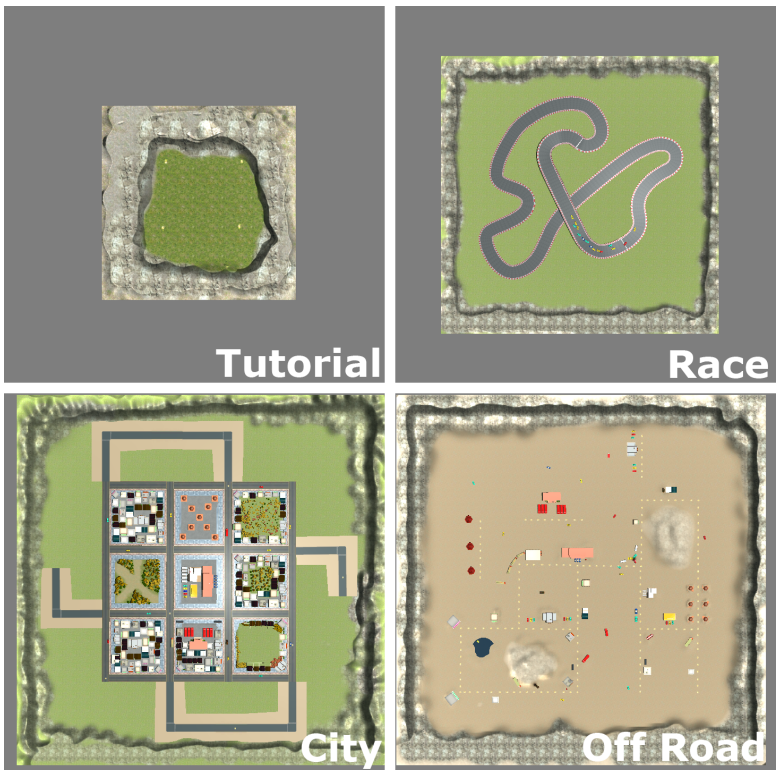


Figure 5.2: Participants tried all four levels. The order was randomly selected (except the tutorial) to avoid potential bias. Note: the tutorial level has been scaled up for the sake of clarity; in reality, it was about four times smaller than the other levels.

5.4 Test Setup

Test participants were placed at a table with an iPad Pro (see Figure 5.4). A facilitator sat next to the test participant with a clear view of the iPad screen and the test participant. The facilitator followed a short manuscript to provide instructions on how to play the game. A notetaker sat behind the test participant (as far back as possible to minimize distractions), while still being able to see what went on in the game. Two cameras recorded footage of the test: one focusing on the iPad screen and the other on the children's facial expressions.

5.4.1 Logged Data

While playing the game, various kinds of data were logged as comma-separated values. Data was logged five times a second. Disregarding the tutorial level, this gave $(120*5)+(120*5)+(90*5) = 1650$ number of samples per participant. The following data was logged:

Figure 5.3: We went out to various schools with a colorful poster. We asked teachers to send it to parents via their intranet as well.

Figure 5.4: The test setup at two different schools.

- Participant ID.
- Time stamp (total seconds).
- Interaction style: *AppMATes* or *Road Focus*.
- Whether the tablet has detected the widget or not.
- Number of touch points (between 0 and 3).
- Current phase (tutorial, city, off road, race).
- Widget position X and Y in viewspace (between 0 and 1).
- If any road is currently visible on the screen (in the off road level, coins are used instead).
- If the player is currently driving on a road — and if so, what type of road (straight or corner/intersection). In off road level, coins define an invisible road.
- Road orientation of the current road (horizontal or vertical).
- Distance to the middle of the road the player is currently driving on, if any.
- Angle between the car's forward direction and the orientation of the road the player is driving on, if any (i.e. between 0 and 90).
- Distance to the current forward screen edge in centimeters.
- Distance to the closest current side edge in centimeters.
- Current forward screen edge (up, right, down, left).
- Angle between the widget's forward direction and a vector from the widget to the closest corner of the screen

5.5 Results

In the following, we present our results from the 64 test participants. Based on the hypotheses mentioned in Section 5.1, we wanted to examine if there are any significant differences between *AppMATes* and *Road Focus*. We did this using Mann-Whitney U tests, which yielded a group of p-values. These are displayed in Table 5.1.

5.5.1 Distance to Forward Edges and Side Edges (Centimeters)

We calculated the average distances to the forward and closest side edge for each test participant. The data was calculated as both an average for all of the phases, as well as each of the individual phases. The median forward edge distance (all phases) for *AppMATes* was 8.35 cm ($M = 8.54, SD = 0.97$) and 8.17 cm ($M = 8.02, SD = 0.80$) for *Road Focus*. The median side

Table 5.1: P-values calculated by comparing *AppMATes* and *Road Focus* using a Mann-Whitney U test with alpha = 0.05. Left-tailed = *AppMATes* > *Road Focus*; right-tailed = *Road Focus* > *AppMATes*.

Average Distance (right-tailed)	**All Phases**	**City**	**Off Road**	**Race**
P-values of Forward Edge	0.9472	0.9964	0.9964	0.9712
P-values of Side Edge	2.4947e-09	1.6286e-04	1.0590e-04	1.8060e-04
Crossing 5 cm Threshold (left-tailed)	**All Phases**	**City**	**Off Road**	**Race**
P-Values of Forward Edge	0.2102	0.8490	0.8616	0.6664
P-values of Side Edge	1.3569e-08	0.0155	0.0071	0.0031
Driving on Roads (right-tailed)	**All Phases**	**City**	**Off Road**	**Race**
P-values of Frequency	0.7719	0.3485	0.9059	0.0452
Off-Center Problems (left-tailed)	**All Phases**	**City**	**Off Road**	**Race**
P-values of Frequency	N/A	0.0519	0.0191	N/A

edge distance (all phases) for *AppMATes* was 7.00 cm ($M = 7.00, SD = 0.63$) and 8.20 cm ($M = 8.20, SD = 0.54$) for *Road Focus*. This data is shown as boxplots in Figures 5.5 and 5.6.

It turns out that there is no significant difference when it comes to the forward edge distances ($p > 0.05$), meaning $H0_1$ cannot be rejected. However, looking at the side edge distances ($p < 0.05$), we can reject $H0_2$. As the *Road Focus* method tries to align the current road with the car's horizontal direction, this makes sense. Because it does not take into account the forward direction of the car, but only the horizontal directions, it only has a significant impact on the side edge distances.

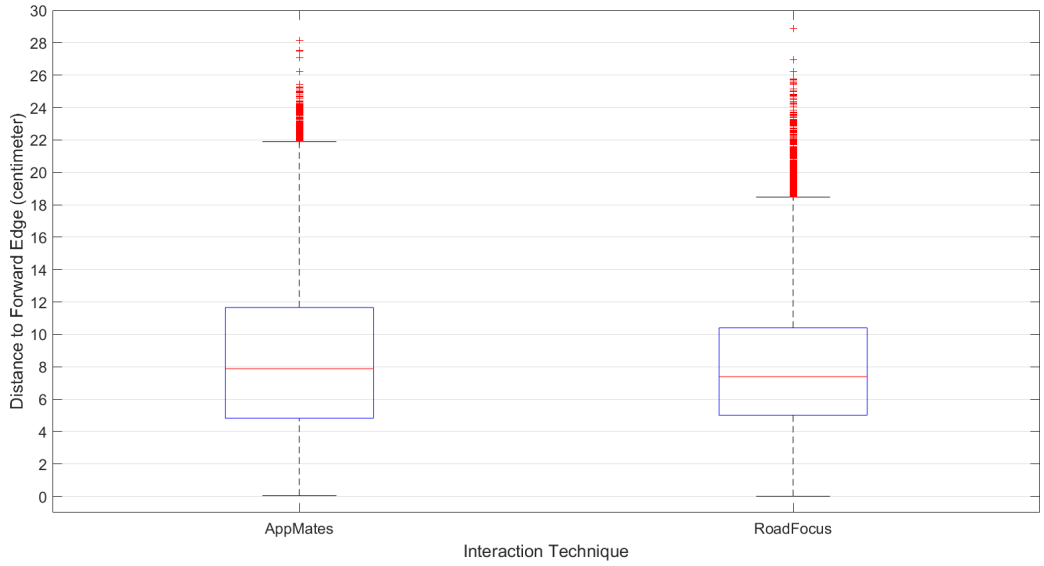


Figure 5.5: Boxplot showing the average distance to the forward edge — all phases (higher is better).

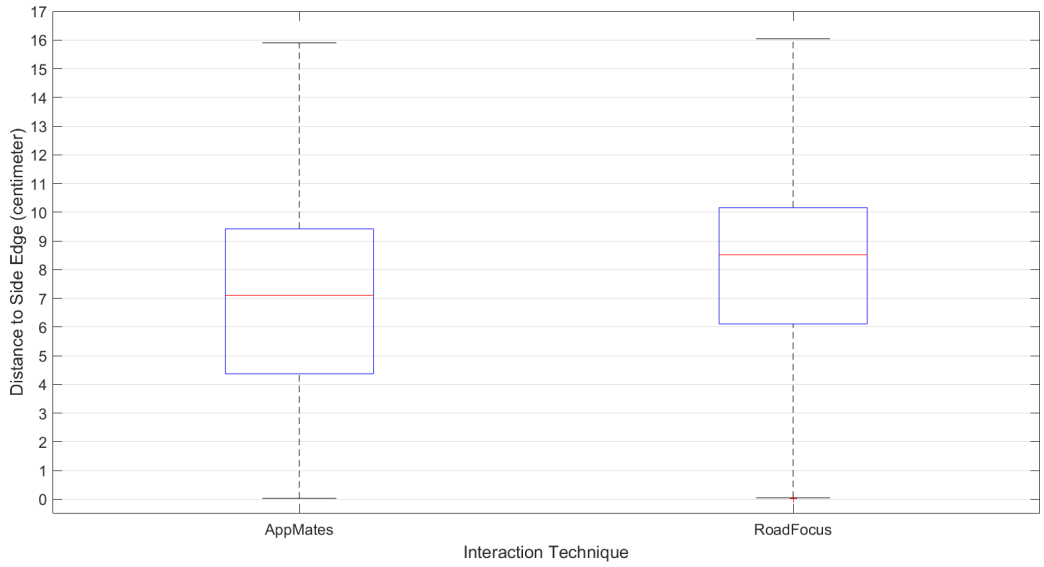


Figure 5.6: Boxplot showing the average distance to the side edge — all phases (higher is better).

5.5.2 Side and Forward Edge Threshold of 5 Centimeters

We set a boundary threshold of 5 centimeters, meaning that if the widget was closer than 5 cm to the edge, we logged and summed it for each participant. We deemed being closer than 5 cm to the edge of the screen was problematic. Therefore, we wanted to see if *Road Focus* resulted in fewer of these problems.

For the forward edge problems, the median sum of occurrences for *AppMATes* was 452 ($M = 438.0, SD = 153.03$), while for *Road Focus* it was 395 ($M = 410.30, SD = 154.91$). There appears to be no significant differences, so we fail to reject $H0_3$. Looking at the side edge problems, the median sum of occurrences for *AppMATes* was 492 ($M = 505.67, SD = 140.87$), while for *Road Focus* it was 266 ($M = 269.20, SD = 105.06$) (or approx. 16% of the samples). The p-values here show that there is a significant difference ($p < 0.05$), so we can reject $H0_4$. This naturally follows the results mentioned in Section 5.5.1.

5.5.3 Time Spent Driving On Roads

We were interested in seeing if *Road Focus* made the participants drive more on the roads than with *AppMATes*. For each test participant, we counted how often they were driving on a road in each of the phases. It turns out that there is no significant difference in most of the phases. Only in the race is there a significant difference ($p < 0.05$). In the racing phase, the

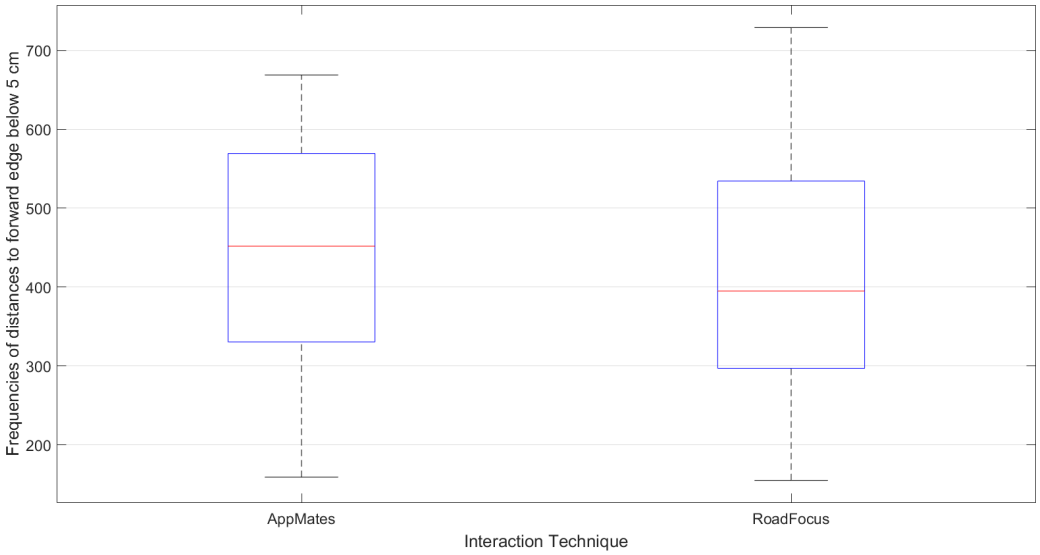


Figure 5.7: Boxplot showing number of occurrences where distance to forward edge is below 5 cm - all phases (lower is better).

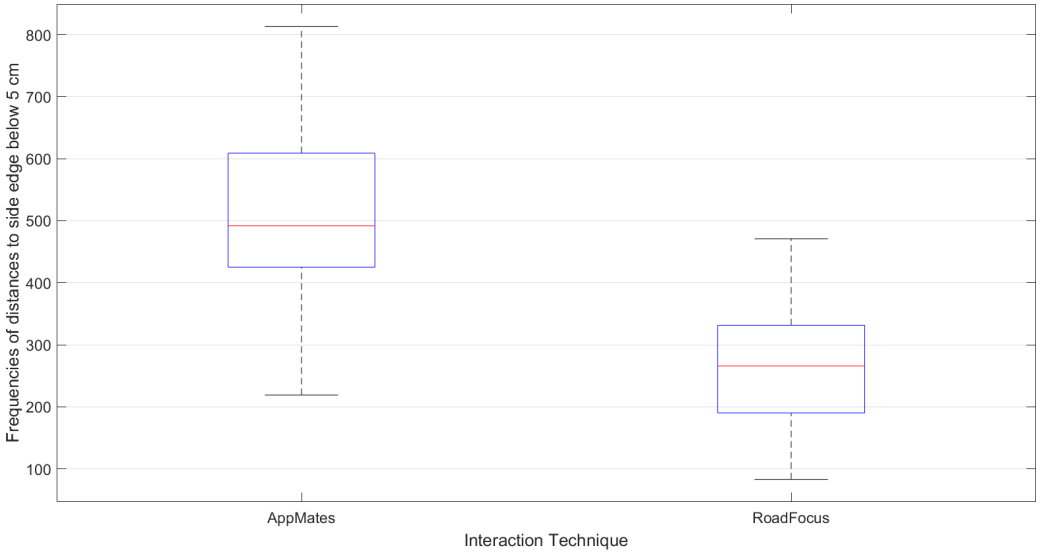


Figure 5.8: Boxplot showing number of occurrences where distance to side edge is below 5 cm - all phases (lower is better)

median frequency of driving on the road is 424.50 ($M = 420.94, SD = 25.39$) for *AppMATes*, and 437.00 ($M = 432.43, SD = 17.40$) for *Road Focus*. Its corresponding boxplot is shown in Figure 5.9. This means that we fail to reject $H0_5$ for all other phases than the race phase. We believe that the reason for this difference in the race phase might be due to situations where the road becomes difficult to follow with *AppMATes*. If a participant drives close to the the road edges and screen edges, there is a possibility the camera will force the player off the road. Instead of trying to get back on the road, many children "cheated" and drove to a different part of the race track. Doing so meant that participants drove off the race track for a short while, hence why they are are driving less on the road with *AppMATes* than *Road Focus*.

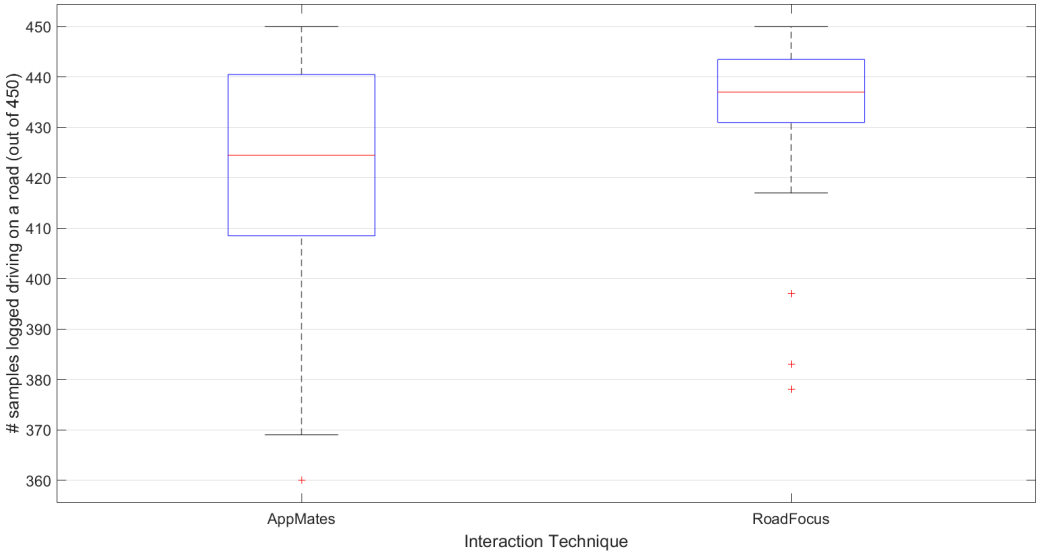


Figure 5.9: Boxplot showing the frequency of how often participants drove on the road in the race phase (each race lasted 90 seconds = 450 samples in total).

5.5.4 Off-Center Road Problems

One problem that often occurred with *AppMATes* was that players drove close on the edge of a road. If this happens while they were close to a side edge of the screen, it could make it difficult to follow the road, since the road might drift further and further away from the visible screen. To spot these issues, we looked at instances where players are *1)* driving on a road, *2)* driving more than 3 in-game meters from the center of the road (almost at the edge of the road) and *3)* driving within the 5 cm side edge threshold of the screen. These problems are shown in Figures 5.10 and 5.11 (note: for practical reasons it was not possible to do this check for the racing level). In the city phase, the median frequency of this problem happening was 5.00 ($M = 6.65, SD = 6.46$) for *AppMATes* and 3.00 ($M = 4.19, SD = 3.89$) for *Road Focus*.

In the off road phase, the median frequency for *AppMATes* was 7.50 ($M = 9.19, SD = 7.23$) and 4.50 ($M = 5.44 SD = 4.46$) for *Road Focus*.

Looking at the p-values, we can see that there is a significant difference in the off road phase ($p < 0.05$), while there is a near-significant difference for the city phase with *Road Focus* ($p = 0.0519$). This means we can reject $H0_6$. This aligns with what we expected: that players tend to drive less along the edges of the screen with *road focus*.

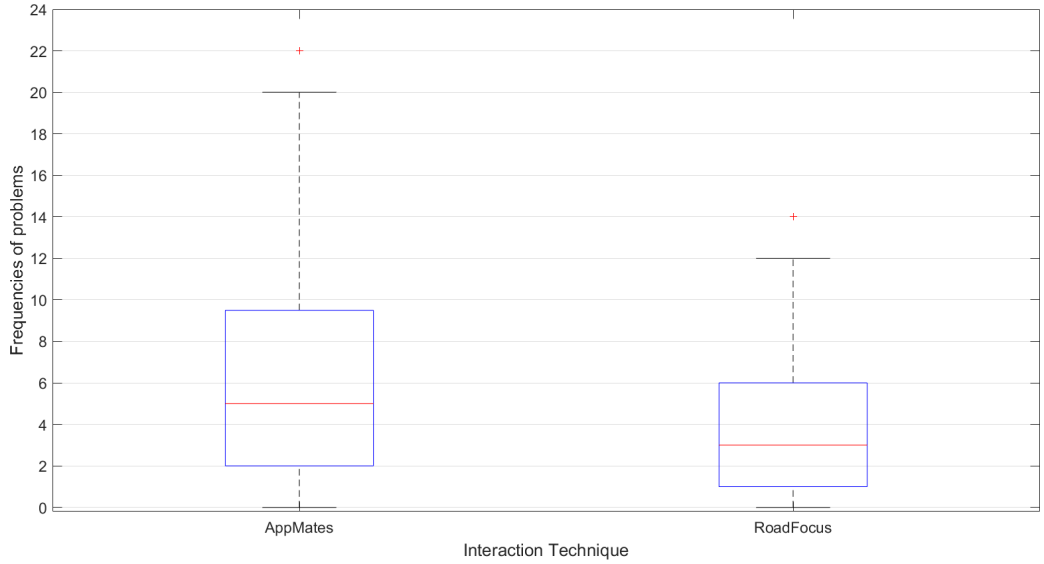


Figure 5.10: Boxplot showing problems with being off-center on a road and off-center of the screen at the same time - city road phase (lower is better).

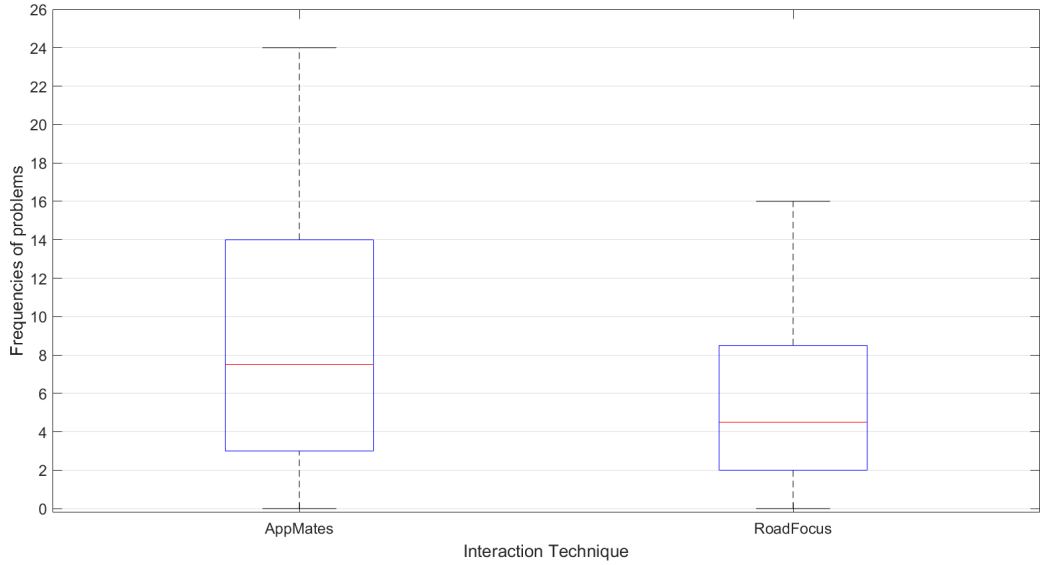


Figure 5.11: Boxplot showing problems with being off-center on a road and off-center of the screen at the same time - off road phase (lower is better).

Chapter 6

Discussion and Conclusion

Developing digital games with tangible widgets presents unique challenges, because both designers and players have to think in two different "worlds", namely the physical and the virtual. Both include their own sorts of affordances and constraints. Looking at *AppMATes*, we saw issues where children would push the widget too close to the screen edges. With this project, we have iterated on many different interaction techniques to counter this problem. During our evaluations, we found most of them to have drawbacks, either related to how intuitive they were to use, or how much cognitive load they required from the children. We also learned that the metaphor for the interaction technique is important. In this particular case, it was vital that the widget felt like a car and that the controls mirrored this.

We can relate the fundamental dilemma in this project to the camera challenges described in Section 3.4. We saw that children had a desire to drive on the roads, and we, as the designers, were interested in them driving in the center of the screen. The challenge was to develop a system that smoothly and comfortably met both of these needs. In the end, we decided to go with a similar approach as *AppMATes*, but added a small tweak on top of it in the form of *Road Focus*.

We can conclude that *Road Focus* was successful in what it was built for. Even though we can see that participants didn't drive more or less on the roads in the city, the participants drove less alongside the screen edges. A problem that often occurred with the *AppMATes* solution was that the participants found themselves alongside the edge of the screen on a road, from which they were slowly driving off. This scenario has also been significantly decreased with *Road*

Focus. The effect of this can, in particular, be seen in the race track level where participants stayed significantly more on the road than with the *AppMATes* solution. However, it didn't help in decreasing the distance to the forward edge.

We learned that it is important to observe and look at how players naturally behave in a game. Instead of forcing them into undesirable behaviours, we should strive towards utilizing their natural ways of playing said game. In the case of them pushing the widget too close to the screen edges, we could have prompted them with some kind of external GUI element to instruct them to stop doing that. However, we instead chose to exploit their intrinsic motivation to follow the roads. *Road Focus* doesn't force, reward or punish players into driving further away from the edge, but instead nudges them in an unobtrusive way that follows their natural predisposition.

Chapter 7

Future Work

If we were to take this project further, there are several aspects we could look into. For example, we could spend more time on analyzing the behaviours recorded in the video footage in both the initial tests as well as the final test. Due to time constraints, we were unable to perform an in-depth video analysis of all footage. This meant that we knew most about the symptom (driving close to screen edges) and not so much the causes for this. It might be possible to get a deeper understanding of *why* the children actually tend to drive so close to the edges. Also, our initial tests consisted of about 20 participants across different ages. This meant that we only had the chance to test each of nine interaction techniques with a few participants. Having a larger sample size in the initial tests might have revealed useful information that could be used for further designs. In our final tests, participants spent about five minutes each. We do not know if the desire to drive along roads (as well as collecting coins) will persist over longer play sessions, e.g., 10, 30 or 60 minutes.

If the children did indeed push the widget close to the edges in order to drive faster, we could look into what would happen if we actually *let* them drive faster by doing this. Building on this, we could investigate which desire is the most prevalent: driving fast or driving along roads.

Acknowledgements

We would like to thank the following for providing us with test participants for this project:

- Aalborg Bibliotekerne
- Filstedvej Skole
- Sofiendalskolen
- Sct. Mariæskolen
- Børnehaven Tusindfryd
- Klostermarksskolen

Bibliography

[Bar00] Joel F Bartlett. Rock'n'scroll is here to stay [user interface]. *Computer Graphics and Applications, IEEE*, 20(3):40–45, 2000.

[BCL12] Dan Burnett, Paul Coulton, and Adam Lewis. Providing both physical and perceived affordances using physical games pieces on touch based tablets. In *Proceedings of the 8th Australasian Conference on Interactive Entertainment: Playing the System*, page 8. ACM, 2012.

[BFTK14] Mads Bock, Martin Fisker, Kasper Fischer Topp, and Martin Kraus. Initial exploration of the use of specific tangible widgets for tablet games. In *Social Informatics*, pages 183–190. Springer, 2014.

[BFTK15] Mads Bock, Martin Fisker, Kasper Fischer Topp, and Martin Kraus. Tangible widgets for a multiplayer tablet game in comparison to finger touch. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pages 755–758. ACM, 2015.

[BKLJP04] Doug A Bowman, Ernst Kruijff, Joseph J LaViola Jr, and Ivan Poupyrev. *3D user interfaces: theory and practice*. Addison-Wesley, 2004.

[Bla11] Andrew Blakney. *Tangible User Interfaces and Metaphors for 3D Navigation*. PhD thesis, Concordia University, 2011.

[Che03] Anthony Chemero. An outline of a theory of affordances. *Ecological psychology*, 15(2):181–195, 2003.

[Cou15] Paul Coulton. Playful and gameful design for the internet of things. In *More Playful User Interfaces*, pages 151–173. Springer, 2015.

[DM02] Afke Donker and Panos Markopoulos. A comparison of think-aloud, questionnaires and interviews for testing usability with children. In *People and Computers XVI-Memorable Yet Invisible*, pages 305–316. Springer, 2002.

[DR04] Afke Donker and Pieter Reitsma. Usability testing with young children. In *Proceedings of the 2004 conference on Interaction design and children: building a community*, pages 43–48. ACM, 2004.

[FB06] Doris Pronin Fromberg and Doris Bergen. *Play from birth to twelve: Contexts, perspectives, and meanings*. Taylor & Francis, 2006.

[FIB95] George W Fitzmaurice, Hiroshi Ishii, and William AS Buxton. Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449. ACM Press/Addison-Wesley Publishing Co., 1995.

[HB06] Eva Hornecker and Jacob Buur. Getting a grip on tangible interaction: a framework on physical space and social interaction. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 437–446. ACM, 2006.

[HLZ04] Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1, 2004.

[Hou08] Juan Pablo Hourcade. Interaction design and children. *Foundations and Trends in Human-Computer Interaction*, 1(4):277–392, 2008.

[HPGK94] Ken Hinckley, Randy Pausch, John C Goble, and Neal F Kassell. A survey of design issues in spatial input. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 213–222. ACM, 1994.

[HSU04] Lars Erik Holmquist, Albrecht Schmidt, and Brygg Ullmer. Tangible interfaces in perspective. *Personal and Ubiquitous Computing*, 8(5):291–293, 2004.

[HTP+97] Ken Hinckley, Joe Tullio, Randy Pausch, Dennis Proffitt, and Neal Kassell. Usability analysis of 3d rotation techniques. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 1–10. ACM, 1997.

[Hyb13] Hybridex. Disney Appmates. https://hybridex.wordpress.com/2013/05/13/disney-appmates, 2013.

[Ish07] Hiroshi Ishii. *Tangible user interfaces*. CRC Press, 2007.

[Juu10] Jesper Juul. *A casual revolution: Reinventing video games and their players*. MIT press, 2010.

[KBNR03] Boriana Koleva, Steve Benford, Kher Hui Ng, and Tom Rodden. A framework for tangible user interfaces. In *Physical Interaction (PI03) Workshop on Real World User Interfaces*, pages 46–50, 2003.

[Ker15] Itay Keren. Scroll Back: The Theory and Practice of Cameras in Side-Scrollers. https://docs.google.com/document/d/1iNSQIyNpVGHeak6isbP6AHdHD50gs8MNXF1GCf08efg/pub, 2015.

[KWRE11] Sven Kratz, Tilo Westermann, Michael Rohs, and Georg Essl. Capwidgets: Tangible widgets versus multi-touch controls on mobile devices. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1351–1356, New York, NY, USA, 2011. ACM.

[Nor13] Donald A Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[Pow99] Thomas G Power. *Play and exploration in children and animals*. Psychology Press, 1999.

[Ull02] Brygg Anders Ullmer. *Tangible interfaces for manipulating aggregates of digital information*. PhD thesis, Citeseer, 2002.

[WW11] Daniel Wigdor and Dennis Wixon. *Brave NUI world: designing natural user interfaces for touch and gesture*. Elsevier, 2011.