Independent degree project – second cycle

Master thesis AV, 30 Credits

Computer Engineering

Development of interactive entertainment system for Din Tur buses

Elijs Dima
Development of interactive entertainment system for Din Tur buses
Elijs Dima

MID SWEDEN UNIVERSITY
Department of Information and Communication Systems (IKS)

Examiner: Mikael Gidlund, mikael.gidlund@miun.se
Supervisors: Ulf Jennehag, ulf.jennehag@miun.se; Björn Lidenmark, bjorn.lidenmark@dohi.se
Author: Elijs Dima, eldi1000@student.miun.se
Degree programme: Intl. Master’s Programme in Computer Engineering, 120 credits
Course: Computer Engineering MA, Final Project (30hp)
Main field of study: Computer Engineering
Semester, year: VT, 2015

ii
Abstract
Public transport organizations, such as Din Tur, require affordable modern solutions to improve their public image and passenger satisfaction. An easy way to create positive associations in people is to provide either useful services or, preferably, entertainment. This report covers the design, development and evaluation of an entertainment system – consisting of a smartphone game, a cloud-hosted backend, and a supporting on-bus hardware system – with the objective of making Din Tur's bus service seem more modern and appealing. The smartphone game, “Håll Platsen”, is developed in Unity game engine, focuses on providing brief entertainment during bus commutes, and incorporates gamification design elements. The Python-based back-end resides in Google's App Engine and Datastore platforms, and provides a unified virtual game environment enabling player cooperation and competition. The prototype on-bus hardware system uses the Raspberry Pi as a Light-Emitting Diode control system to supply real-world feedback of the game's virtual environment. The systems incorporate real-world busstop positioning, player location, online mapping services, team location-control mechanics, reflex-based minigames, player progression mechanics, and mobile-focused design. The resulting system can be useful in estimating public response to non-standard “smart” promotion methods, the use of games to improve everyday routines (i.e. commuting), and serve as a basis for further research in human & smart-technology interaction.

Keywords: Entertainment systems, Public transport, Mobile, Din Tur, Unity, Game development
Acknowledgements

The author would like to thank the employees of Din Tur, Dohi Agency AB, and Björn Lidenmark in particular, for providing informative and technological support for conducting this thesis.

Additional thanks to Ulf Jennehag at Mid Sweden University for supervising this work and providing much-appreciated insights in deployment of physical prototypes and project evaluation procedures.
# Table of Contents

Abstract .................................................................................................................................................. iii
Acknowledgements ................................................................................................................................. iv
Terminology ........................................................................................................................................... vii

1 Introduction ......................................................................................................................................... 1
  1.1 Background and problem motivation ............................................................................................... 1
  1.2 Overall aim ........................................................................................................................................ 1
  1.3 Scope ................................................................................................................................................ 2
  1.4 Concrete and verifiable goals ............................................................................................................ 2
  1.5 Outline ............................................................................................................................................. 2
  1.6 Contributions ................................................................................................................................... 3

2 Theory ................................................................................................................................................ 4
  2.1 European Bus System of the Future ................................................................................................. 4
  2.2 'Smart' technology and public perception ....................................................................................... 5
  2.3 Din Tur ............................................................................................................................................ 6
  2.4 Scalable databases ............................................................................................................................. 6
  2.5 Google App Engine .......................................................................................................................... 8
  2.6 Cross-platform mobile development tools ....................................................................................... 8
  2.7 Unity Game Engine .......................................................................................................................... 9
  2.8 Gamification in context of Games .................................................................................................... 11

3 Methodology ....................................................................................................................................... 13
  3.1 Development process ....................................................................................................................... 13
  3.2 Testing & Evaluation ......................................................................................................................... 14

4 Pre-development plan ......................................................................................................................... 16
  4.1 Client-side application plan ............................................................................................................... 16
  4.2 Back-end system plan ....................................................................................................................... 18
  4.3 Hardware system plan ....................................................................................................................... 18

5 Client-side Application ......................................................................................................................... 19
  5.1 Game scenes & User Interface ......................................................................................................... 19
  5.2 Geolocation & Mapping system ...................................................................................................... 22
  5.3 Minigame ....................................................................................................................................... 23
  5.4 Player progression & storage .......................................................................................................... 25
  5.5 Script process flows ......................................................................................................................... 26

6 Back-end System ................................................................................................................................. 29
  6.1 Structure ........................................................................................................................................ 29

7 Hardware Installation Prototype ......................................................................................................... 32
  7.1 System structure .............................................................................................................................. 32
  7.2 Process lifecycle ............................................................................................................................... 34

8 Development Changes ......................................................................................................................... 36
  8.1 Game Design changes ...................................................................................................................... 36
Development of interactive entertainment system for Din Tur buses
Elijs Dima

8.2 Application Interface Changes..........................................................37
8.3 Backend/feature changes..................................................................39
8.4 Lighting system changes....................................................................39

9 Results....................................................................................................40
9.1 Application network resource use......................................................40
9.2 Application performance.................................................................41
9.3 App Engine server resource use.......................................................43
9.4 User test results................................................................................45
9.4.1 Online survey responses..............................................................45
9.4.2 Individual test results....................................................................48

10 Discussion............................................................................................50
10.1 Result overview................................................................................50
10.1.1 Networking performance............................................................50
10.1.2 Back-end functionality...............................................................50
10.1.3 Application performance...........................................................50
10.1.4 Application user feedback..........................................................51
10.1.5 Hardware system functionality...................................................52
10.2 Goal completion.................................................................................52
10.3 Ethical aspects..................................................................................52
10.4 Usefulness........................................................................................53
10.5 Future work.......................................................................................54

References................................................................................................55

Appendix A: Test survey forms...............................................................60
  Public test invitation form.................................................................60
  Public test feedback form.................................................................61
  In-person test feedback form.............................................................68

Appendix B: Application Network Test....................................................70
  Use case definition & data gathering method......................................70
  Single test run procedure.................................................................70
  Numeric recorded data........................................................................71

Appendix C: Application Performance Test............................................72
  Measurement Terms............................................................................72
  Measurement Method........................................................................72
  Measurement Script..........................................................................73

Appendix D: Post-test game changes.......................................................74
  Minigame changes.............................................................................74
  Map-screen changes..........................................................................76
  Text/Grammar changes.....................................................................76
  Light-system changes........................................................................76
Terminology

Acronyms
(order of appearance)

EBSF  European Bus System of the Future (*pan-european research initiative*)
EU  European Union
IT  Information Technologies
LED  Light-Emitting Diode
WLAN  Wireless Local Area Network
GPS  Global Positioning System
UITP  *Union International des Transports Public* (French: International Union of Public Transport)
DB  Database
SQL  Structured Query Language
RDBMS  Relational Database Management System
BASE  Basically Available, Soft-state, Eventually-consistent
NoSQL  Not-only SQL
UX  User eXperience
UI  User Interface
PHP  *Recursive*: PHP Hypertext Preprocessor
API  Application Programming Interface
URL  Uniform Resource Locator
HTTP  Hyper-Text Transfer Protocol
PaaS  Platform as a Service
FPS  Frames Per Second
Development of interactive entertainment system for Din Tur buses
Elijs Dima

1 Introduction

1.1 Background and problem motivation
The day-to-day life is becoming increasingly digitized, and seemingly non-technical aspects of it are trending towards merging into the Internet-of-Things and the Social Internet culture. Smart-homes, smart-appliances, smart-cars are becoming trendy and popular, and the general public tends to respond well to innovations along these lines. Likewise, the personal smartphone (and to a lesser extent, personal tablet) has become a practical necessity for the vast majority of people in Sweden.

Din Tur, the public transport agency in Sweden's Västernorrlands län, is looking for ways to make use of the aforementioned tendencies to improve the public perception of their company and services, and to increase the use of their public transport services by enticing new customers to become Din Tur's passengers, and by motivating existing, regular passengers to use Din Tur's services more frequently. Currently, their transport infrastructure (trains, buses etc.) is not inherently making use of the smart-technology trend, and despite regular improvements to their web-services, there is no strict way of making people associate the physical transport vehicles and bus stops with modern technologies. An increased interest and utilization of public transport may lead to long-term gains in sustainability and ecological impact via reducing the use of personal transport.

1.2 Overall aim
The project's aim is to design, implement and test an interactive system that is deployable in Din Tur's public transport vehicles and makes use of the prevalence of people's personal smart-devices and their common use as time-spending entertainment devices during regular commutes. The system, once deployed, has to be enticing and interesting enough to make passengers use it, and possibly generate media interest, and ultimately lead to increased public approval of/interest in Din Tur, and possibly an increase in use of Din Tur's public transport services (as measured by Din Tur's internal departments). The system must also be relatively easy to interact with, and not require overly large effort to use on the passengers' part. The result of this project should be useful to Din Tur for indicating the future prospects of improving their infrastructure, making use of modern solutions for service promotion & customer interaction, and the public's likely reaction towards such practices. Hopefully, the result shall also directly lead to further implementation of the developed system, or an improved version thereof, across a wider range of public transport vehicles in Sweden.
1.3 Scope
The study is focused on developing a functional prototype that satisfies the specified problems and works towards motivating people to use Din Tur public transport for travelling. The development shall take financial (maximum budget allowance of <20’000 SEK) and practical (one-person development team, Feb-May timeframe) limitations into account, and the resulting system shall be deployed on at least one of Din Tur's buses for purposes of evaluation and user-testing. The results and observations of peoples' interaction with the prototype shall be documented and represented in this report, within bounds of public disclosure and within permitted limits of any non-disclosure agreements associated with this work.

1.4 Concrete and verifiable goals
The concrete goals of the project are as follows:

- Research the possibilities of using “smart technology” as promotional / entertainment material during everyday tasks (specifically, commuting), and associated theory on viable platforms, relevant technology, existing research.

- **Design/plan** an entertainment system/application for use whilst commuting by bus.

- Develop the **client-side application** (game) of that system, using Unity (Game Engine) for development. Possibly apply Gamification principles during feature design.

- **Develop back-end system** to support the front-end application.

- Develop **prototype** for a **hardware installation** to be used on buses to support / enrich the game system.

- **Test** performance of client-side application and back-end system.

- Conduct public user **tests** to evaluate client-side application (with functioning back-end), in-person and remote sessions.

- Deploy the hardware-installation on public service bus(es), and perform a public launch of the game in cooperation with Din Tur.

- From project results, **evaluate** the feasibility of using similar systems for promoting public transport infrastructure providers.

1.5 Outline
Chapter 1 introduces the general problem/task and sets the context for this report. Chapter 2 provides a brief overview on the current state of “smart technologies” in terms of public perception, the European bus-manufacturer initiatives for modernizing public transport, the current state of Din Tur, and the technolo-
Development of interactive entertainment system for Din Tur buses

Elijs Dima

1.6 Contributions

The thesis has been performed by Elijs Dima, under supervision of Björn Lidenmark (Dohi Agency) and Ulf Jennehag (Mittuniversitetet). Din Tur has provided the resources for prototype deployment and testing, and Dohi Agency has provided workspace and some of the equipment used for developing the system. Likewise, Mid Sweden University has provided equipment for conducting early prototype development. All system development, artwork, design and programming has been done by the author.
2 Theory

This section shall contain all information required for the reader to understand the context and technologies associated with this study (assuming a medium existing level of understanding of modern technologies).

2.1 European Bus System of the Future

The European Bus System of the Future (EBSF) is a four-year pan-European cooperation between 11 European Union (EU) countries and five leading bus manufacturers; the project was conducted between 2008 and 2012, with the resulting findings made publicly available [1]. The stated aim of the EBSF initiative was to develop a new generation of urban public transport system that is more attractive to the public, and makes use of new technology developments and infrastructures. The EBSF project involved design of new transport vehicles (fig. 1), research of adaptations of computer and 'smart' technology, infrastructure development and operational practices.

EBSF's research in Bremerhaven, Germany, included deploying a number of Information Technology (IT) systems on demonstration buses, to examine public response to on-board e-public services [2]. The demonstrator buses made use of Light-Emitting Diode (LED) lights to illuminate door chains, a seat occupancy indication system that uses coloured lights above seats for indication, an on-board wireless local area network (WLAN) router, and a Global Positioning System (GPS) amplifier for real-time location look-up. The bus also contained a number of digital signage displays (19” - 20”) located throughout the interior to display real-time departure timing and traffic status information for passenger benefit (fig. 2).
From the use-case tests in Bremerhaven, EBSF concluded that the use of informative illumination and display systems led to a “new look and feel [3]”, and made the buses seem “cool” and “sexy” [3]. They also note that there was a sense of information overload among passengers, and that there is a need to consider future trends such as “Passenger 2.0” and “fast-moving mobile techniques”[3]. The results also indicate that the largest potential is on focusing on services, not devices, and the integration of technological innovations such as location-based services, augmented reality, and “app”-like services.

The general findings were that the implementation of EBSF use cases in Madrid, Paris (related to improving the bus-stop designs with digital display systems) and Bremerhaven resulted in an increased perception of the general quality of bus services (by 7%). ESBF also notes, however, that socioeconomic factors may negatively influence public transport service demand and use (citing the economic crisis as the main factor for a 10% reduction in service demand) [3].

The EBSF initiative is being continued by a follow-up project, the “intelligent, innovative, integrated Bus Systems project” (3iBS), which started in October 2012 and is scheduled to end on March 2015 [4]. The 3iBS initiative is coordinated by the International Association for Public Transport (UITP).

2.2 'Smart' technology and public perception

A study on smart grids in residential contexts, by Stragier et al [5], determined that in day-to-day context, the intention of people to use smart technologies is influenced by their perceived usefulness and the peoples' attitude towards technology; attitude, in turn, is mostly influenced by perceived ease of use and perceived usefulness. Within a smart-household context, Stragier et al found that the main challenges on enticing people to use smart systems lie in the loss of control and increased complexity that new, unfamiliar devices may bring.

In their analysis of smartphone application users, Verkasalo et al (see [6]) posit that social norms have a positive effect on perceived enjoyment and usefulness of new technology. They re-affirm that perceived technical incompetency is a significant hindering factor in adoption/use of new software, and that both perceived enjoyment and usefulness both contribute to a person's intention to use a given technology. Verkasalo et al note that for non-users in particular, perceived

Figure 2: Signage displays on EBSF Demo Bus. 
Source:[3]
usefulness is less of a factor than perceived enjoyment, especially in context of contacts with their peers, implying that enjoyment perception is the target for motivating non-users to adopt a technology. In the specific context of mobile game applications, perceived enjoyment was “the key variable” [6]. Their conclusion is that for existing smartphone owners are likely to use their smartphones as the primary devices of interaction with new applications/services, owing to the perception of control and familiarity with the physical device. Moreover, despite objective compromises and downsides in smartphone capabilities for a person's individual lifestyle, most smartphone users exhibit a positive attitude towards the devices, and are willing to use applications even if they are flawed [7].

A study on public transport user experience enhancement via interactive technology, conducted by Foth and Schroeter, notes that “the act of traveling on public transport usually remains a rather dull experience, which most commuters would not associate with fun” [8]. The task of entertaining the passengers is usually left in their own hands, with smartphones being a particular source of individual entertainment [8]. The prospect of sharing a regular route with 'familiar strangers' is identified as a common occurrence in public transport that forms an impromptu social network, and has potential for facilitating interactions and interest between passengers; the authors also advocate the inclusion of the transport vehicle itself as a key factor in systems that enhance public transport user experience (such as equipping buses with digital guest-books, or by using the bus/route as a connection locator).

2.3 Din Tur

As several studies world-wide indicate (such as [9], [10] and [11]), from an environmental perspective public transportation is more sustainable than private transport – public transportation is less energy intensive and produces less carbon dioxide (CO₂) by an order of magnitude, and is statistically safer in terms of accidents [9]. In Europe, urban and suburban buses service a total of 56% of local public transport journeys [12], making bus services the most prominent of public transport types.

In Västernorrlands county, Sweden, public transport is managed by a public transport agency, an association of the county’s seven municipalities; this initiative is commonly known as “Din Tur”. The public transport authority's self-stated goals are to make strategic decisions about public transport, to procure and finance services and conduct operational activities in terms of national mobility, school transport and public transport for the participating municipalities, and coordinate inter-regional commuting [13].

2.4 Scalable databases

Computing system scalability can be defined in multiple ways, such as speed-up, efficiency, isoefficiency, isospeed. A practical way of estimating scalability for cloud-deployed systems is to measure to performance change during workload change [14]. Based on such metrics, it was established that horizontal scal-
Development of interactive entertainment system for Din Tur buses
Elijs Dima

2015-06-12

horizontal scaling (spreading across multiple machines) resulted in greater performance than vertical scaling (increasing single machine's processing speed/power).

Commercially-deployed databases (DBs) nowadays are largely split in relational management systems (commonly based on Structured Query Language (SQL)), and non-relational database systems (see Fig. 3), which allow for compromises in data consistency [15] to allow horizontal scalability and read/write speeds more suited for Big-Data and Big-Audience cases [16], resulting in a BASE system type (Basically Available, Soft-state, Eventually consistent).

![Figure 3: State of NoSQL databases in real world, ca. 2011. Source: [15]](image)

Relational database management systems (RDBMS) are able to scale vertically (by extending the individual processing power of the servers at exponential cost), and to scale read-accesses horizontally, by implementing a master-slave server system (this may create replication lag in the order of minutes or more, however) [17]. A further option is to partition the database across multiple servers (‘sharding’), requiring an additional management layer to determine the destination of writes.

In contrast, Not-only SQL (NoSQL) databases are non-relational and designed to scale horizontally [18]. Common NoSQL databases are based on key-value pairs, document-stores, graphs or wide-columns. In social game contexts, document-based NoSQL DBs are more commonly used (according to [16]), because they can provide low-latency data access, consistency whilst moving data between servers, and schema/query flexibility without losing the ability to perform sophisticated queries.

7
2.5 Google App Engine

Google App Engine [19] is a cloud-based application hosting service, designed on the Platform as a Service (PaaS) principle. It provides the benefit of being deployed on Google's infrastructure, and abstracts server management / maintenance issues. Applications for this platform can be written in Java, Python, PHP Hypertext Preprocessor (PHP) and Go, which can all integrate with other Google-maintained cloud services / Application Programming Interfaces (APIs). Applications are sandboxed and isolated from server hardware/OS/location concerns, and are restricted from writing to the local filesystem or interfering with other applications. The Python Runtime is the faster of the two available non-Beta environments, and includes a simple web application framework (webapp2) by default, with optional support for larger third-party solutions (such as Django). Scaling is automated, with application instances being launched and dismissed based on incoming request demand and allocated platform quotas.

Data storage in App Engine applications can be implemented in a third party database, or in Google's Cloud SQL and Datastore Databases. Cloud SQL is a mySQL database hosted on Google servers. Datastore is a schema-less NoSQL document store, with zero planned downtime, transaction atomicity, and high read/write availability. CloudSQL is stated to be appropriate for small-medium applications, whereas Datastore is advertised as a preferred solution for scalable/distributed applications [20].

Applications deployed on App Engine platform are charged based on Instance running-time, incoming bandwidth, datastore operations, log size and application size, with quotas calculated on a daily basis. A certain amount of resources are assigned as free quotas, and are deducted from the application's quota consumption. Without a billing contract, an application exceeding the free daily quota of any resource will be blocked from further use of the resource for the remainder of the calendar day [21].

2.6 Cross-platform mobile development tools

Developing applications/games for mobile platforms may occur in two ways – by developing platform-native code for every deployment platform, or by using cross-platform development tools (CPT). The main reasons for opting to use a CPT are the increased development speed and multi-platform support features.
Many CPTs rely on using web-development as the interface of choice, allowing developers to create web-applications wrapped in the platforms' web-window containers. This approach is not as convenient for mobile game development, as games tend to have greater necessity for low-level hardware interaction and native API access, and benefit from specialized, game-oriented functionality and development tools/environment. The fast pace of mobile platform evolution has led to a sparseness in objective evaluations of development tools; however, most recent available articles (such as [23], [24], [25], [26]) all identify Unity [27] as one of the preferred engines / development environments for mobile cross-platform game development.

2.7 Unity Game Engine

Unity is a fully-featured game development engine, that allows for 2D and 3D scenes. It contains an internal animation system, physics engine, renderer/game controller, and abstracts all hardware / platform OS-level controls. A Unity game consists of one or more 'Scenes', which contain 'Game Objects' – data entities that may be rendered and have a specified position in the game's virtual 3D space ('world space' or 'game world'). A game object can be a 3D mesh, a 2D sprite, a UI element, a logical structure with child game objects, a camera, etc. Game objects have 'Components', which are attachable and define the object's properties, behaviour and interactions. Common components are a 'Transform' (controls the object's location & scale), 'Renderer' (controls how the object is rendered to screen), 'Collider' (allows the object to react with Unity's
physics engine), 'Script' (C# or JavaScript code with regular & special functions) and any others (see [28] for further information).

The "main game loop" is not directly accessible/editable. Instead, all scripts have a specific lifecycle, which interacts with the game's engine. Control to a script is passed intermittently, via special event functions (such as 
\([\text{Awake}()]\), 
\([\text{Start}()]\), 
\([\text{Update}()]\), 
\([\text{ FixedUpdate}()]\), 
\([\text{onDisable}()]\) - see fig. 5 and [29]). It is possible to call public functions between scripts, and to adjust public variables directly, and it is possible to spin off functions in parallel execution paths (via 
\([\text{Coroutine}()]\) command); it is not possible to directly affect the multithreading, performance, and execution process of the "game thread" itself.

For retrieving content from Uniform Resource Locators (URLs), Unity contains a 'WWW' utility module [31], which supports GET and POST requests and web-form data generation. The module is intended for lightweight, non-constant data transmission/retrieval in cases where an authoritative server or direct multiplayer functionality is not required (such as managing high-score lists). Author-
itative servers have to run simulations or instances of the game for each player, to validate and control their state changes; this introduces a server resource overhead that may be unnecessary for single-player games, thus making the WWW alternative more optimal. To avoid blocking, in-game network requests have to be called within co-routine functions, which contain appropriate yield points for pausing execution until a relevant event/state change.

User input is abstracted via the Input interface [32], which encapsulates keyboard, touchscreen, accelerometer, mouse, and other common input device signals. Most key/touch-based abstractions contain 'pressed', 'released' and 'pressed-down' phase transition flags. During script execution, these interfaces can be checked for the relevant properties to see the event phase, as well as information about event position on-screen and phase duration (if applicable). Inputs are commonly polled during the game's “Update()” phase, to allow for frame-fast response at the game's fastest supported rate. User input does not produce events/interrupts as such, so it is possible to plan the execution order of user input handling.

Since the “frame” - the cycle of processing game input, simulating AI and game objects, updating the world and rendering the visuals – is a fundamental game concept, performance of Unity games is defined in terms of Frames per Second (FPS), which inversely correlates to seconds-per-frame [33] [34] [35]. Performance optimizations and measurements are directed towards achieving a stable baseline perceived FPS number, which gives the user an impression of smooth performance. (Certain aspects of the game may run at a slower rate – the perceived FPS/smoothness is related to the graphical performance, not necessarily the underlying game script calculations). A common target performance for games is 30 FPS, which approximately corresponds to cinematic and televised frame update rates that users are already used to. Common methods for ensuring a high perceived framerate are hiding resource-intensive tasks in separate threads / behind static loading-screens, splitting tasks into segments across multiple frames, and decoupling visual feedback (animations, transitions) from data processing.

As a multi-platform environment, Unity allows to export projects on a Web-Player platform, and native Windows / Linux / OSX / iOS / Android / Windows Phone / BlackberryOS platforms.

2.8 Gamification in context of Games

As commonly agreed by field researchers (e.g. in [36], [37], [38]), 'gamification' is generally defined as the use of game design elements in non-game contexts ('game' in context equates to 'computer/video/mobile software game'). However, studies and definitions of gamification help crystallize those elements of game design that specifically entice people to interact with the gamified application.

As G. Zichermann, a prominent pro-gamification activist and early promoter of the concept, puts it, “gamification [allows] marketers to focus on what they know best – convincing consumers to take loyalty and purchasing actions – us-
ing a powerful toolkit of engagement gleaned from games” [41], [42]. This exposes numerous ethical problems that require consideration when interacting with gamification, as well as the notoriously non-specific, “buzzword-like” quality of the term – as game designer and scholar I. Bogost points out (ref. [41]), using the term “exploitationware” instead of “gamification” would convey the same underlying development choices and design practices without the linguistic bias imposed from “gamification” make-up. (Disclosure: the author of this report generally agrees with I. Bogost's stance on 'gamification', and the exploitative problems it introduces to software development. Please refer to sources referenced in this subchapter for alternate views on the issue of gamification ethics).

Nevertheless, the fact remains that current research in gamification – from scholars and investment-focused consultant-promoters alike – is related to extracting the 'enticing elements' from game design. Therefore, by specifically incorporating those elements in a game, there is a higher likelihood that people will react positively and have higher engagement.

Common elements cited by multiple 'gamification' studies are the use of personalized progression – contextualized in terms of user achievements, badges, levelling mechanics; and repeat-incentivized gameplay – implemented via time-limited or difficulty-gated gameplay sections, time constraints, repeating “turns” of game-time ([36], [38]). Additionally, three behavioural needs for user's motivation to accomplish a task are highlighted ([37]): relatedness, the need to interact and connect between multiple people; competence, the need for being effective and mastering a problem in any given environment; and autonomy, the need to control one's own life. The responsible use of these elements, and game design decisions that support the outlined needs, may lead (by adapting Zichermann’s citation) to “gamified games” - games with intentional consideration of game engagement toolkit use for purposes of consumer loyalty and purchasing action encouragement. An irresponsible (or unethical) use of those same elements could lead to development of blatant “exploitationware”.

Development of interactive entertainment system for Din Tur buses
Elijs Dima 2015-06-12
3 Methodology

The project consisted of two main stages – the system development stage and the system deployment / testing stage (both stages were preceded by a short initial system planning/design stage). The development was expected to take roughly 3 time units, while the deployment/testing stage would take 1-2 time units to conduct (assuming a total of 5-6 time units for the entire project, with one unit dispersed throughout for documentation and ancillary task scheduling; 1 time unit may be a month or more – these are given as estimates of the relative distribution of work involved, not as a strict time-plan of the project).

3.1 Development process

The development phase has consisted of an incremental build model with free-form (due to single-person development team) feature-driven development focus. The overall feature groups were defined as “Basic game operation”, “Interactive map functionality”, “User interface”, “Minigame functionality/styling”, “Player Progression”, “Backend integration” and “On-Bus Light System”, with appropriate mini-features within each group (see fig. 6 for mid-development feature trace). To save on administrative overheads, and to allow for new systems/platforms learning, there was not a formal development process plan – instead, feature groups were worked on in a progressively layered fashion, with mid-development project control sessions with Din Tur representatives, designers/developers at Dohi Agency, and test group(s), to adjust and redirect development.

The control sessions with Din Tur can be categorized in 'client meetings' with primary contact-persons, 'test sessions' with Din Tur's selected test groups, 'deployment meetings' for handling issues with hardware and software deploy-
development, and 'followup meetings' for evaluating the final outcomes of the project from the company's perspective. Within Dohi Agency, design demonstration, evaluation and planning sessions ('workshops', 'pitch presentations') were conducted to gather additional ideas and feedback on design, UI/UX and features. The aforementioned control sessions were be used to direct the development process (in terms of features, timing, deliverables). The outcomes / results of the sessions would be aggregated and presented in a separate chapter in this report, in order to document the mid-development changes. Additionally, individually-directed test sessions were organized as necessary, depending on the development progression.

The developed system/hardware would be in a “deliverable” state when the components could function independently in a natural environment and provide the basic intended experience to the user. The three main states of feature deliverability were: - an Isolated Mobile Application/Game (IMA) that handles locally-stored progression, game-play & map interaction; - a Connected Mobile Application (CMA) that supports full IMA functionality with addition of backend-server support for globally-coordinated play/leaderboards/busstop states; - an Augmented Mobile System (AMS), where the CMA experience is bolstered by the On-Bus Lighting System on select Din Tur buses. The IMA-CMA-AMS sequence represents the overall project milestones and development phases (not withstanding bug-fixes, priority changes and emergency tweaks).

3.2 Testing & Evaluation

The final evaluation of the delivered system would consist of multiple segments.

The server backend, if delivered, would be evaluated on request responsiveness, speed and resource consumption, based on metrics taken from the chosen system's management interface. Performance during pre-deployment and deployment phases would be considered (respectively, light and heavy load).

The metric-based results would serve to show the functionality and capability of the system, and would be useful in informing future work, as well as act as a point-of-comparison for similar / competing projects and applications. The user & company-based feedback results would be useful in estimating the resultant impact of the deployed system, and to inform any other project of similar scope or focus that aims to collaborate with non-academic institutions.

The Mobile application / game was evaluated on performance metrics (frame-rate / responsiveness / network-request timings / loading-times etc.) and feature completion checklists, on user-impressions / feedback, and on Din Tur's feedback.

The mobile game user-testing consisted of remote, larger-scale public-access tests with feedback gathering via online surveys (ref. Appendix A), and a smaller set of individual test sessions with interested respondents. The test demographic was not specifically restricted, other than by means of social
Development of interactive entertainment system for Din Tur buses
Elijs Dima
2015-06-12

reach/availability, as the application is loosely targeted towards multiple age groups (~14 to ~40+ years). Gender and other personal questions (besides age) were omitted from the surveys to protect user privacy and reduce the survey length (in order to make it less intimidating to start).

Invitations to participate in public remote testing were sent to interested parties among university students and staff, volunteer participants from Dohi Agency, and pupils from Kramfors public school. From these sets, volunteers were selected to participate in more extensive, individual task-oriented test sessions. Participants of both test types were asked to perform a set of basic 'tasks' to accomplish in the application. The feedback given by participants, as well as observed trends/behaviours from individual tests, would be quantified and presented in this report. Feedback was divided into overall impressions, specific impressions of game sections (initial login/registration, interactive map, minigame), feedback on technical issues, and optional additional commentary on further improvement suggestions (see appendix A for full survey form).

Individual tests were performed with respondents from the Mittuniversitetet student and faculty populations (due to expedience and availability). During the procedure, test-users were instructed to accomplish a list of objectives, whilst their decision-process and actions taken towards accomplishing those steps was observed. Following the objective-driven test, feedback was solicited within several categories (see appendix A for list of questions), to obtain detailed impressions and personal perspectives from the participants. The participants would be given help with the initial tasks of the test (installation of the game on their devices, initial registration) only if overly large problems are encountered, and only after the participants' own attempts have ended. No help would be given for the other objectives, unless explicitly asked-for. All such difficulties (if any occur) would be explicitly marked as “Did Not Finish” in the objective accomplishment category, with indications towards the point of failure.

The hardware lighting system would be evaluated on functionality aspects – a checklist of features it needs to support to provide basic functionality (such as being able to control LED lightstrips, being able to access the backend via internet, being able to read GPS coordinates and ascertain its own position, being able to compare it's own coordinates with known busstop coordinates). Due to delays in coordinating between multiple involved agencies, the light-system on-bus deployment did not take place before this report's finalization, and therefore has not been documented here.
4 Pre-development plan

This section will report the initial system design, that was created after the background study, but before any tests or meetings with Din Tur. The purpose of this section is to describe the starting-point of the system development, and to let the reader identify the impact of ongoing control sessions (ref. ch. 3.1) on the project.

4.1 Client-side application plan

The game will be developed using Unity, a cross-platform game development engine/IDE. The game will be based on the idea of team-oriented 'area capture', with areas being the Din Tur bus stops – the collaborative aspect fulfilling user's relatedness needs (see ch. 2.8); the general concept is similar to the ideas behind the games Ingress [39] and Turf [40]. Whilst on a bus, the players will have the chance to play a mini-game (with estimated length of a minute, to account for bus travel times between stations, and time-limiting by progressively incrementing difficulty – to fulfil users' competence need and repeat-incentivized gameplay) and gather highscores and 'team influence' per each Din Tur bus stop (personalized progression). The team with highest influence per station will “own” it and, in game-mechanics terms, provide extra bonuses towards individual player highscores. When not on a bus, the game will still show the bus station map, but not allow the players to play the score-influence-gathering minigame (to reinforce the connection between the game and the bus service).

Therefore, the 'game' application will consist of two main parts – an interactive map (see fig. 7) showing geo-located bus stations and their assigned 'game' statistics, and a playable mini-game (see fig. 8). There will likely be additional game sections for User Experience (UX) benefit, such as an info/help screen,
login/registration page (with possible social-network login), player settings etc. (see fig. 9 for an initial estimate of program flow from user perspective).

Figure 8: 'Minigame' pre-development visual concept

Figure 9: Game transition flow (pre-development visualization)

The crucial aspects of the 'game' will be the ability to display online-fetched maps from a map provider (such as Google Maps or MapQuest), to place game-world objects on top of the map in accordance to station GPS coordinates, to determine if a player is on a bus whilst playing, to connect to a support server
for sending and receiving game-state updates and possibly player data, to run a mini-game that allows players to contribute to their teams, and to support player-distinguishing aspects such as usernames, login data, team selections, achievements/progression, highscores etc.

4.2 Back-end system plan

The supporting server system will store the 'game state', in terms of bus station locations and attributes (owner teams, highscores per station), player profiles (progression, team selection, highscores) and team specifics (owned stations, assigned players etc.). The server system will have to have the ability to scale horizontally, to allow for the possibility of fast popularity / userbase growth of the game; for that purpose, a NoSQL database is the preferable choice, possibly deployed either on Din Tur's own servers or on a large-scale cloud provider (Azure, Amazon, Google or similar).

4.3 Hardware system plan

The on-bus lighting system will be deployed in a small number of 'demonstration' buses, and consist of one or several small-size computers (Raspberry Pi) with internet and GPS access, connected to a number of coloured LED strips (see fig. 10). The light elements will provide ambient illumination that correlates to the nearest station's “owner team's colour”, to provide an interesting physical connection for the players between the game 'world' and the real-world buses. The mini-computers will drive the LED strips and provide accurate colour information, sent from the support server based on the bus's GPS-indicated location in relation to the in-system bus stations.

Figure 10: Illumination system pre-development plan. Top-left: bus installation location. Top-right: LED strip with Arduino controllers. Bottom-left: Raspberry Pi / LED strip connection.
5 Client-side Application

This chapter describes the “final” game version at the conclusion of this project, that was subjected to user-testing. For intermediate development steps, and differences between this version and the initial plan, see chapter 8.

5.1 Game scenes & User Interface

The mobile application, built as a Unity game project, is spread across three scenes, which represent the three distinct activity 'screens' that the user interacts with ('Start screen', for registration/login purposes, 'Map screen', for showing the virtual world with geolocated bus stations, and 'Game screen', for handling the playable mini-game). This separation allows to conserve system memory and processing power, and creates logical points (loading screens) where large assets can be unloaded from memory without concern for user-interaction responsiveness. For each scene, the user interface and game scripts are separate and do not interact amongst themselves; there is a specific game object shared between the three scenes, that handles data storage and transferral.

User interface (UI) for all three scenes is built in Unity's internal UI system, unityUI, and consists of individual UI elements (such as panels, buttons, text-boxes) arranged in a Canvas container (proprietary term, not an HTML Canvas), with width, height, anchoring and layering specified in relation to a reference resolution (the canvas handles UI scaling for other resolutions). Interactivity for UI elements is controlled by Unity, which abstracts the OS-
level input events to Unity-level mouse & touch events. UI elements are be animated (see fig. 13) with Unity's animation system (Mecanim), in the same way as any other game object. The animation system is based on animation states (fig. 12), between when transitions can be triggered from scripts, or scheduled to occur automatically; the animations consist of parameter changes over time, as defined by keyframes and progression curves. For UI element appearance/disappearance, more prominent elements have nonlinear animation curves (fig. 14), with rapid initial phase and gradual deceleration for appearance/expansion (and vice-versa for disappearance), as suggested by Google in Material design guidelines [43].

The 'Start screen' UI (fig. 15) allows user to log in / register, to select their team and to view brief help-information. The transitions between any interface state (i.e. between different information displays) are animation to create a more 'game-like', responsive feel. The user has to explicitly press a 'play' button to progress to the next scene, as the scene transition can involve a >1s application freeze (abstracted via a loading screen).
The 'Map screen' scene (fig. 16) contains mesh elements across which the map images (textures) are painted, sprite objects that represent the bus stations and are placed in the game space by translating the GPS coordinates to game scene coordinates (see chapter 5.2), and UI elements for displaying information across multiple panels. As before, UI transitions are animated, with some elements mapped from screen-space to world-space for indication purposes.

The 'Minigame' scene (fig. 11, 19) contains several sprites that form the game's background, foreground, and interactive elements, as well as UI elements. Foreground game objects also have unity physics components attached, to allow for collision-triggered event calls. The scene contains several particle emitters (which emit optimised sprites with individual movement mechanics), one of which is used to show player's touch interaction.


5.2 Geolocation & Mapping system

'Map screen' scene contains several game objects that work to show the player's location and nearby bus stops ('stations') on a map, using GPS data for positioning. To achieve this, the scene uses a (heavily modified version of a) Unity third-party plugin, “MapNav” [44]. A 2D plane mesh contains a game script component, which is responsible for handling the map positioning and displaying logic; this script communicates with, but is separated from, the scene's “main” control script, which handles most of the state data & interaction functions.

On scene load, the script obtains the phone's current position using the built-in GPS sensor, and sets those coordinates as the game world's origin point. Another script is cleared to obtain the GPS coordinates of known bus stops (from locally stored data – a server db iteration over all 2500+ bus stops to get their coordinates would not be well-performing, and waste server resources/quotas), and positions 'station' sprite objects in appropriate places in the game world, above the map plane, by performing conversion between decimal latitude & longitude coordinates, origin point coordinates, and Unity's world coordinates. If the origin point is not within 'known' rectangular coordinate space (e.g. origin point is outside of a major city), a web request to an online mapping service is issued, and a .jpg image of a map tile centred on the origin point, at fixed elevation is returned, and applied as the map plane's texture.
By default, the OpenStreetMap MapQuest service is used, with minor changes to request structure, other online mapping services (such as Google Maps) can be used (the choice of OpenStreetMaps was mainly motivated by terms of service, not technical considerations – other mapping solutions largely prohibit from wrapping the map tiles in custom interfaces, or removing their own mapping service functionality – waypointing, address-searching etc.). The downloaded map tile is low-resolution (640x640px), so for high-intensity areas, such as major regional cities, 4K tiles are pre-loaded with the application; if the origin point is within known boundaries, the high-resolution image is used instead.

The map plane object also captures touch events to detect users' panning/swiping input. The input translation is passed towards the scene camera, which repositions itself and any child objects accordingly. Upon reaching the edge of the current map tile, the camera movement is clamped, and a UI icon is displayed to the user, allowing them to swap to the adjacent map tile (this prepares user for a possible delay whilst new map tile is downloaded, and saves on mobile internet usage).

5.3 Minigame

After initial user-interaction tests, the minigame idea was reworked from the “player controls a running person in infinite runner style game” towards an easier-to-play “player taps away oncoming obstacles” mechanic. The game difficulty is adjusted by tweaking the obstacle forms, movement speed and spawn density, and the player is scored primarily on the number of obstacles removed.

The 'obstacle' objects are composed of multiple individual sprites, each of which contain a trigger area. These objects are instantiated on game start, and are controlled by an object pooling script in order to avoid expensive game-ob-
ject creation/deletion operations [45]. The obstacle objects are instead given a semi-random motion & speed, and are activated on one side of the screen. They have a self-timer, which deactivates the object after a while, allowing it to return to the pool. An external script controls which objects are taken from the pool at what times and with what parameters.

The user interaction during gameplay is handled by a game script that captures user touch input events in every frame; upon touch event, it updates the position of a particle emitter (pink neon streaks for contrast against background elements) for visual feedback, and on a touch-release event performs a depth-oriented raycast from the input position. The raycast triggers any sprite with trigger areas it crosses, allowing the obstacle sprites to 'dissipate'. There are two varieties of obstacle sprites – the 'oncoming cars' and the 'pollution smoke' shrouding them; to remove a car obstacle, first the smoke sprites need to be removed (via tapping on screen – a tap translates into Unity touch events, which have distinct start/during/release phases, which can be checked for during the game's frame update function). Non-removed car sprites can collide with the bus sprite on the left side of the screen, and reduce player 'lives' amount. Each collision also temporarily reduces the game's time factor, providing a “slow-motion” effect, and activates a 'collision' particle emitter over the bus sprite for a more immediate visual feedback.

Several layers of pooled backdrop sprites (see fig. 11) are used to provide the illusion of bus movement, by having a constant motion (with artificial parallax effect from different movement speeds at different layers). Like the obstacles, the background sprites are position-reset once they leave the camera projection area, to avoid deleting/spawning objects and improve the game's performance. For simple elements (such as clouds or shrubbery), additional particle emitters are used. The UI is minimal during gameplay, to reduce the amount of draw calls to the device CPU, and to allow user a larger interaction canvas. At game-over, a full-screen UI panel is revealed with performance metrics and options to restart the game or submit the score, and the game's time factor is zeroed to stop any further object motion, generation and particle activity (the alternative, disabling all active elements, introduces a performance spike for no practical benefit). Player performance is categorized into a “score” metric, which – with behind-the-scenes balancing and normalization – improves the player team's ratings on the given busstop. To provide instant feedback and link the scoring systems with busstop takeover mechanics, a visual, animated transition is played to show the team position changes as a result of player's game performance.
5.4 **Player progression & storage**

To entice repeated play and to provide long-term player motivation, player metrics are stored (locally and on a server). The two individual progression motivators from user perspective are personal achievements and player 'levels'. The levelling system is based on the aggregated 'influence' metric (which, in turn, is closely derived from minigame score result), divided into progression levels based on a 'powers-of-two' division. The achievements are single-unlock data points, and represent various in-game states (e.g. reaching a specific 'score' amount in the minigame). Locally, the progression metrics, along with other persistent data, are stored in serialized binary format on device filesystem, with user having the ability to “clear” their progress (deletes the binary save file). The binary file is not used to persist data between scenes during a single play session, but is instead used to persist data between separate play sessions on the same device. The locally-stored data also serves as a fall-back in case of backend communication issues.

The achievements are controlled by a two-list system, which distinguishes between 'new' and 'unlocked' achievements. Player actions cause trigger an achievement identifier to be added to the 'new' list, from any game scene (duplicates and already-'unlocked' achievements are prevented from being added). Once the application transitions into the 'Map screen' scene (or at any time within it), all 'new' achievements trigger an appropriate UI animation, and are semi-permanently stored (locally and on server) as 'unlocked', thus preventing duplication. Achievement definitions are stored separately and can be expanded at will, however their triggers need to be inserted into the game code.

Player progression data is also stored on the game's back-end server, with two discrete 'upload' points – upon the initial load of the “Map Screen” scene (fig. 16), for synchronizing the achievement-unlock specifics, and at the end-game screen in the “Minigame” Scene (fig. 19), for submitting the game results. The data is sent via HTTP POST-requests via Unity's WWW interface. Progression data downloads occur on game start, after 'login' operation on “Start screen” scene (fig. 15). Thereafter, the data is kept in sync via upload-request responses/results.
5.5 Script process flows

Each of the three Unity game scenes contains a number of game objects with their own associated scripts, that plug into Unity’s script lifecycle function calls (see fig. 5). Instead of defining each and every object's each and every component, it is more practical to consider each scene's interactions with outside factors – the network requests (to backend and external servers), and the user input activities (including data from device sensors, such as the GPS module).

The Start Screen scene's process flow is straightforward, and can be summarized in terms of “show user login interface, tell server to login/register user, show user team selectors, load next scene” (see fig. 20). The server-version request is there at the start to ensure that connectivity works, and, if necessary, cause App Engine to allocate a new instance.

![Start screen process flow (simplified)](image)

Figure 20: Start Screen scene progression. Does not cover detailed feedback / error fallbacks, animation details/triggers, function multitasking/time-delaying for readability purposes.
Development of interactive entertainment system for Din Tur buses
Elijs Dima 2015-06-12

Map Screen scene (see fig. 21 for process progression) involves the most amount of simultaneously-active objects, though most of their activities execute as a result of input events or other user-caused changes. This is the only scene where game scripts interact with third-party servers, for downloading mapping service tiles and for resolving the device's public IP (a fallback method of determining if the player is connected to a Din Tur bus WLAN).

![Map Screen process flow diagram]

The 'Minigame' scene (fig. 22) contains an object pooler that re-uses obstacle objects, spawning them in bounded-random intervals continuously. To stop the spawn activity, and to cease passive-running particle emitters (and other ongoing game processes), the game's relative time rate is set to 0 at start and end of the 'game'. That avoids a potential performance spike that could otherwise occur from disabling objects / explicitly stopping scripts. The time rate is global, so upon every transition to a different scene, the value is restored to default. Frame-rate performance is more important in this scene compared to the other two, as here the user has to react to game's prompts in a timely manner, whilst a significant amount of moving objects and particles is displayed on screen. Therefore, nearly all script activity during gameplay is trigger-initiated, with the game's Update() script sections restricted to user input handling and certain object horizontal translations. Raycasting (for detecting if user tap is 'above' an obstacle object) in particular, as a resource-intensive action, is only launched on touch-
release input event phase (ref. ch. 2.7 for more information). Any server or local data storage/lookup operations occur after the gameplay section (i.e. after “game over”) to conserve resources.

Figure 22: 'Minigame' Scene process flow. Does not cover detailed function multitasking/delaying, animation triggers, particle emission management, object pooling behaviour control.
6 Back-end System

This chapter describes the supporting back-end system in brief detail.

6.1 Structure

The backend system (fig. 23) is deployed on Google's App Engine platform in order to allow rapid development and automatic scaling on demand. The application, based on “Connect Google App Engine” library/sample by L.Young [47], consists of handler functions that are linked to specific URL patterns in the web request. Each handler, written in Python 2.7, serves a specific function (e.g. player login, player registration, obtaining data about busstop nr.XYZ). The handlers interact with the web request object (which contains functions to send response, obtain authentication / session headers, POST data etc.) and the game-specific data in Google's Datastore. The responses sent back by the handler functions are not HTML-formatted, because they are intended to only be called from within the game, outside of a browser context. Each request-response is specified to return as much information as required by the game-event source, to reduce the amount of subsequent requests. Handler scripts make use of Google App Engine API, webapp2, ndb and python-domestic frameworks/libraries.

The data is stored in Datastore, defined by two main data models (busstop and user). The models do not have a strong relational link; the handler methods are tasked with keeping information sufficiently consistent. User data contains login information, user-progression data and user identification data (authentic-
Development of interactive entertainment system for Din Tur buses
Elijs Dima

The system is managed by webapp2_extras.appengine.auth class, providing password salt/hashing. In case of database breach, user passwords are not compromised, and there are no explicit links to other web services. The busstop model contains a station identifier, descriptor (bus stop name), and containers for busstop-specific player leaderboards and team scores. To accommodate the Västernorrland region, the datastore has data on 2159 busstops in total (after eliminating duplicate entries with identical coordinates).

<table>
<thead>
<tr>
<th>Busstop statistics</th>
<th>Entities</th>
<th>Built-in Indexes</th>
<th>Composite Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last updated: 3 hr 59 min ago</td>
<td>Total size: 948.42 KB</td>
<td>6.6 MB</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Entry count: 2,159</td>
<td>62,611</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Average size: 450 B</td>
<td>111 B</td>
<td>–</td>
</tr>
</tbody>
</table>

Breakdown by Property and Composite Indexes

![Busstop data model statistics in App Engine Datastore. Majority of storage space occupied by auto-generated lookup indexes that App Engine creates to speed up simple search query performance.](image)

Figure 24: "Busstop" data model statistics in App Engine Datastore. Majority of storage space occupied by auto-generated lookup indexes that App Engine creates to speed up simple search query performance.
Development of interactive entertainment system for Din Tur buses
Elijs Dima

Figure 25: "User" data model statistics in App Engine Datastore. Screenshot taken during pre-release tests; total number of users within system is, therefore, still low.
7 Hardware Installation Prototype

7.1 System structure

The light-system consists of 6-meter-long LED strips (5050 analog RGB strips, with IP65 waterproofing), controlled by a Raspberry Pi microcomputer. The Raspberry Pi monitors the current system location using a GPS breakout board and antenna, to recognize when it enters the location of a bus-stop. Once triggered, the microcomputer uses a WiFi antenna module to request colour-information from the game server (using the bus' on-board internet hotspot). Based on the response, a PWM-modulated control signal is passed to power transistors, to regulate the voltage for each colour channel on the LED strips. The lights are powered directly from a 12V 12A power source, whereas the power going to the microcomputer is changed to 5V using a switching step-down buck converter. This power is passed to the micro-USB charging port, to make use of Raspberry Pi's built-in surge-protection systems.

![Diagram of lighting system plan](image.png)

Figure 26: Lighting system plan
Figure 27: Lighting-system functionality test (end of May, 2015)
7.2 Process lifecycle

The application on the Raspberry Pi is written in python, and is initiated on system boot. It consists of two main execution lines – the location-polling thread and the (main) location comparison thread, with the main thread setting up required variables, and launching the polling thread.

![Diagram of process lifecycle]

The location-polling thread is responsible for constantly fetching the latest GPS lat/lon values from the serial interface (using gpsd daemon) and writing them to a global variable. This thread needs to execute with a high frequency, because the data-stream from the GPS module is serial, and all position updates form a queue that needs to be constantly emptied (otherwise, prolonged operation leads to buffer overruns). Therefore, this thread has the bare minimum functionality, to allow it to run at high iteration rate. It contains brief exception handlers, that, in case of exception, try to stop the gpsd daemon, create a new gpsd connection to the serial connection, and re-initiate itself after a brief sleep timeout.

The main location-comparison thread matches the latest-known system (bus) position with a list of known busstop coordinate/identifier pairings. If the current location is within x meters of the closest busstop (value of x can be fine-tuned after deployment, depending on the real-world GPS performance; gener-
ally, a 35-meter distance radius would be sufficiently large to allow for the GPS module reading error. The list of iterated busstop coordinates is reduced to less than 300, to only cover the city of Sundsvall and the immediate surrounding areas. The positioning check is performed once every 4 seconds, to allow for reduced processor usage. In each check, if a new busstop is detected as being within range (which qualifies as “bus passing the busstop”), a web-request is issued towards the backend server, sending the busstop identifier and expecting a returned colour identifier. This, instead of polling the webserver on every location-check, allows to conserve the server operating costs. The server's returned colour determinant is then passed to the GPIO pins via the pi-blaster library, which allows to set fractional values for the R, G and B control gates. The library makes use of a memory controller to act as a pin PWM controller; this does not influence the CPU workload, and frees up computational resources for other operations. In case of an un-predicted exception, the LED light control pins are set to 'off', the gps-polling thread is stopped, and the program restarts from the beginning.
Development Changes

This section covers the major changes in the development process between the initial idea (see ch. 4) and the final system (see ch. 5, 7) that resulted from development sessions (ch. 3.1). A major change in this context is defined as a diversion from the initial plan, or as a full restructure of specific elements between two functional states; non-implemented / non-designed aspects are not included in this overview.

8.1 Game Design changes

The primary gameplay mechanic has changed from infinite-scroller 'running man' type game (fig. 29) to a 'remove obstacles from bus' path' minigame (fig. 30), due to an early play-session, which highlighted two problems – the 'running man' concept did not accommodate low-skill players (high barrier to entry), and did not provide a sufficiently strong thematic link between the game and Din Tur's public transport services.

Obstacle-removal systems started out as a player swipe-tracker, wherein parts of an obstacle are removed as player swiped over them (initially devised as geometric-figure-based tree objects). Design sessions and brief player feedback indicated the need for more thematic consistency, and less fragmented / small-detail-focused interaction, more suited to small mobile device displays. The obstacles were therefore changed to represent oncoming, pollution-emitting cars, with “pollution” shown as a persistent smoke cloud over each car.
Based on Din Tur's feedback towards demonstrated versions, as well as design workshop sessions at Dohi Agency, the visual style of the game has moved from silhouette/shadow-based sprites (f.29) to brightly coloured, 'cartoonish' elements (f.31, f.33) that are easier to read on a small device screen and are less evocative of an “American western-like desert setting”.

The scoring mechanics were simplified as part of larger interface modifications, with the aim of reducing scoring complexity and making player's contribution towards bus-stop team ownership more obvious.

8.2 Application Interface Changes

The first interface versions for the Map-scene used transparency-based panels and detailed information areas on an active panel, a bottom info-bar, and a side-panel (fig. 36). Following test-person feedback and Unity engine specifics, a more Material-like visual style was chosen instead, with information separated into a 'map view', 'player info panel', 'station info panel' and 'help panel' (fig. 37 - 39). The four-colour scheme was transitioned from Unity's default-colours to Google's Material-design suggested colours, and later tweaked to match the colours used in Din Tur's logo.
Based on hands-on testing, the overall amount of information density was reduced via removing behind-the-scenes information, simplifying the scoring/bonus system, removing a “daily challenges” planned feature, and increasing the relative size of remaining information fields, to better support 4”-5” smartphone displays. The 'player info panel' contents were divided into distinct 'progression' and 'achievements' segments for readability purposes, and transitioned to a more icon-depended categorization.

The game start-scene interface was continually updated to fit with the reworked schemes, particularly in terms of component animations and state transitions. After a session with Din Tur representatives, the “no team” option was added as a valid play mode, to allow players to delay the decision until after seeing the game mechanics in action.
8.3 Backend/feature changes

An early session with Din Tur specified that the backend system could be deployed on the company's servers, and user login details could be appropriated from the existing Din Tur web store system. After later sessions, it was decided to instead use a third-party cloud for backend deployment, and dismiss the web-store account integration due to API-development overheads on the company's IT side.

8.4 Lighting system changes

The first version of the light system was designed around two separate power sources, for separate 12V and 5V supply for the LED strips and Raspberry microcomputer, respectively. Following a technical meeting with engineers at 'Stjarnafyrkant' (responsible for performing the installation procedure to not void the bus warranty), the lighting system plan was adapted to function from a single 12V power source. Likewise, the optional plan for obtaining GPS data from bus' existing on-board systems was dismissed as infeasible.

![Figure 44: Two-power-source plan](image1.png)  
![Figure 45: Single-power-source plan](image2.png)
9 Results

9.1 Application network resource use

Data download metrics tracked by logging `WWW.downloadedBytes` parameter in Unity on every request (“RQx”), and adding the HTTP header length (20Bytes, observed via Postman REST client) * Nr. of requests. Performed two sets of multiple tests, with player-location set within a city, and outside of one. See Appendix B for test activity procedure and logged data. Web requests are regular HTTP GET/POST requests towards the App Engine / mapping servers.

Figure 46: Application network-request behaviour in “typical” use case (ref. Appendix B). GPS simulates user being within a city with built-in map tile. RQ1-9 connect to App Engine system, MFRQ connects to MapQuest servers. Sub-Chart: Nr. of request calls (per run per request-type).

Figure 47: Application network-request behaviour in “typical” use case. GPS simulates user being in location without built-in map tile. Sub-chart: Nr. of request calls (/run /type).
9.2 Application performance

To measure sustained framerate (Frames Per Second – FPS – measured as 1/frameDuration), a custom logging script was added to a data storage object that persists across all scenes. The script recorded the timestamp, current scene, and time-since-previous-frame for all frames during the test (see appendix C for script/measurement details). Tested on mid-range 2013 Android smartphone (Sony “Xperia SP” C5303; dual-core 1.7GHz CPU, 1GB RAM, Android v4.4). Additional tests performed on an Apple iPhone 5s, with similar performance.

![Frame performance sequence](chart)

Figure 49: Frame performance sequence, actual (blue) and linear-trending (bright green line), with active-scene ID. Chronological, not time-scaled.
Figure 50: Frame performance distribution in FPS, average-smoothed (FPS per frame = \(\text{Avg}(\text{preceding 2 frame FPS, current frame FPS, following 2 frame FPS})\)) to eliminate single-frame imperceptible spikes. Chronological, not time-scaled.

Figure 51: Time-scaled, chronological, smoothed frame-performance distribution (ref. fig.50 for smoothing specifics). Includes trending-framerate (blue line with drop-shadow).
9.3 App Engine server resource use

Metrics taken from App Engine application management console during user-test phase (May 8\textsuperscript{th}, 2015). Request URI's reworded to “RQx” form to match with fig. 46, 47, 48.

<table>
<thead>
<tr>
<th>URI</th>
<th>Requests/Minute</th>
<th>Requests Last 24 hours</th>
<th>Runtime MCycles Last hour</th>
<th>Average latency Last hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>R05</td>
<td>1.6</td>
<td>103</td>
<td>9</td>
<td>38 ms</td>
</tr>
<tr>
<td>R06</td>
<td>0</td>
<td>31</td>
<td>31</td>
<td>390 ms</td>
</tr>
<tr>
<td>R07</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>157 ms</td>
</tr>
<tr>
<td>R03</td>
<td>0.2</td>
<td>17</td>
<td>2</td>
<td>20 ms</td>
</tr>
<tr>
<td>R01</td>
<td>0</td>
<td>15</td>
<td>22</td>
<td>144 ms</td>
</tr>
<tr>
<td>R02</td>
<td>0.2</td>
<td>6</td>
<td>23</td>
<td>92 ms</td>
</tr>
<tr>
<td>R04</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>R08</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>R09</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 52: Application Request latency metrics

Figure 53: App Engine backend traffic, 2-day period. Does not include Map-service requests made from application to map-service provider servers. Spiked periods correlate with application test sessions.

Figure 54: App Engine backend instance activity (Nr. of active instances averaged per second), 2-day period.
Figure 55: App Engine / Datastore Memcache traffic, 2-day period. Memcache is the intermediate layer between the Datastore and App Engine, that allows for subsequent automatic caching of Datastore entities.

Figure 56: App Engine / Datastore Memcache size auto-adjustment, 2-day period.

Figure 57: App Engine / Datastore Memcache effectiveness (hit ratio), 2-day period. Flatlining at 0% coincides with no activity from application side.
9.4 User test results

9.4.1 Online survey responses
Response options ranging from Bad-Great are scored on a 5-point scale (1 = Bad, 3 = OK, 5 = Great), and per-question score is a sum of individual respondent scores. See appendix A for survey forms.

Figure 58: Online Survey, Question 1: Rate the login/registration part of the game

Figure 59: Online Survey, Question 2: Rate the map/busstop part of the game
Question 5: “Indicate any technical problems you think you encountered”. No technical problems indicated by any respondent.
Development of interactive entertainment system for Din Tur buses
Elijs Dima

Figure 62: Online Survey, Question 7: Play-count categorization

Figure 63: Online Survey, Question 8: Would you like to play this game again sometime later?

Figure 64: Online Survey, Question 9: Respondent age categorization
9.4.2 Individual test results

Due to the observational & conversational nature of the individual tests, this section will document the minor and major factors revealed in each test category. (Minor factor: More than one respondent encounters this; Major factor: more than 50% of respondents encountered this). 2/3rds of all respondents tested the application on their own smartphones, 1/3rd tested on a Sony Xperia SP. Test participants: 4 university students, 2 university staff members. Test process: Ask user to fulfil a task, without instructing how to. Observe user interaction with the application, noting any unexpected interactions, complications, or emotionally-involved actions. If user struggles with completing a task, note the fact, and provide instructional assistance in completing so that the test can proceed. After all tasks are accomplished, ask the user to provide overall feedback and their perspective on the application.

Game start, registration, logging in, team selection stage:

- Minor factor: Pre-written placeholder email (“din@email.se”) causes confusion – not all testers deleted it or noticed it when typing their own address.
- Minor factor: Password is not stored – this is not expected behaviour for some testers

Minigame playtesting stage:

- Major factor: Game mechanics (i.e. repeated tapping on cars) is not clear. Some testers need more than one session to figure it out.
- Major factor: All testers manage less than one minute on first game session.
- Major factor: Most testers go back and play more game sessions after the first one without prompting.
- Major factor: Feedback indicates game difficulty is high, but appropriate.
- Major factor: Comments that game feels very smooth, fluid.
- Minor factor: Results screen is not clear enough. Points aren’t clearly connected with busstop influence piechart.
- Minor factor: Dark UI does not match with the rest of the game colour-wise.
- Minor factor: Testers estimate that minigame might get a bit repetitive after some time. Several indications that extra lives, bonuses, or other, non-car things to tap on are expected.
Map-screen testing stage:

- Minor factor: Achievements menu is not instantly recognizable to some testers. The 'star' icon is not representative enough.

- Minor factor: confusion between personal total (cumulative) point score and per-game highscores.

- Major factor: some testers did not try to open the help window, but clicked on other areas instead to test functionality.

- Minor factor: some testers remark that map panning does not feel entirely responsive.

Post-test general feedback:

Visual style: Commonly positive response. A couple of remarks indicating that icons could be more prominently used.

Performance: Universally smooth.

Minigame gameplay: Often described as simple – sometimes as positive- sometimes as negative, depending on the tester.

Ease of Use: Generally described as clear and easy to interact with.

Interesting-ness: Some remarks about minigame needing more/different types of obstacles or bonuses.

Technical issues: One tester managed to skip minigame results screen due to rapid tapping.

Overall: Positive remarks, with some notes on minigame requiring more diversity.
10 Discussion

10.1 Result overview

10.1.1 Networking performance

The networking measurements (ch. 9.1) indicate that, by multiple orders of magnitude, the main potential consumer of mobile data rates under common game use are the map-tile download requests (fig. 47), which both take time and download capacity. In comparison, the other connections between the game and the backend are minimal, and well under kilobyte sizes (f. 48). It must be said, however, that the intended app use scenario is with a player being on a bus, with their phone linked to the bus' open-access WLAN – therefore, such a scenario should not incur any mobile data charges to the end-user. Likewise, within a city, map tiles are not downloaded from servers at all (f. 46), leading to less network utilization and no possibility of large mobile data-charges.

10.1.2 Back-end functionality

On back-end side (ch. 9.3), it is evident that the requests execute within reasonable practical timings (in the order of milliseconds, without crossing the half-second mark, see f. 52). Moreover, request activity in turn causes the Memcache to be updated to speed up possible subsequent requests (f. 56). Granted, the Memcache hit rate (f. 57) is overall low – this is due to the test procedure not invoking that many repeated requests. In post-release state, the interaction of many users requesting similar data at once ought to lead to better Memcache hit rate. The App Engine instance scheduler appears to be pro-active in stopping instances, leading to a start-stop scenario (f. 54). This, again, is expected to be fixed by a larger number of concurrent users keeping the instance “in-demand” for longer consecutive periods.

10.1.3 Application performance

Performance-wise, the application performs very well, with a stable average & trending framerate of 60 FPS (f. 51). Individual frame metrics (f. 49) indicate frame-spiking, but a nearest-neighbour average smoothing shows that on a human-perception level, the framerate largely remains focused around 60FPS, with very few dips into 30s (f. 50). There are some noticeable gaps in performance (f. 51) – most prominently, on loading a new scene and at about 3 seconds after loading Scene nr.2 (correlating figures 50, 51). New scene loads are hidden by static loading screens, to hide any framerate issues from the user. The latter issue coincides with “revealing” the scene with map and n-hundred busstop objects for the first time – the addition of so many active elements temporarily stresses the game engine's renderer. This has been practically solved by fading the loading-screen out instead of removing it. That way, the moment the loading screen becomes non-100%-opaque, the rendering object allocator kicks in, whilst – to the user – the loading screen still remains practically opaque (254/255ths opaque, to be exact), and the performance-drop is not noticeable. Throughout all other activities, the application retains a smooth, high framerate,
Development of interactive entertainment system for Din Tur buses

Elijs Dima

2015-06-12

even when network activity or animation-intensive tasks are performed. The feedback from test users also supports these observations.

10.1.4 Application user feedback

With remote user tests, there was a very low respondent turnout (and a comparably low number of installs). This may have been caused by a security “feature” in Android's Chrome browser, that prevents downloaded .apk application installation files from being launched, even if the phone's security settings explicitly allow the installation (behaviour not observed on Android 4.4 devices, has been observed on several Android 5.0 devices). This does not effect the published version, as that relies on Google PlayStore, but may have caused a reduction in test install numbers. Functionality post-install is not affected by this.

Statistically-derived knowledge extraction is complicated with such a low respondent amount, therefore the online-survey results likely to be only indicative, not definitive. Overall, though, specific aspects can be observed. Overall, the feedback seems to be positive, generally being located between “Quite good” and “Great” for nearly all rating-based parameters (fig. 58, 59, 60, 61). The reported willingness to play the game once it's released (fig. 63), or at a later stage, also supports the generally positive reception (as does the relatively high play-count metric from fig. 62). Detailed demographic evaluation has not been conducted on this dataset due to (unexpectedly) low respondent amount; any demographic measures at this point would be misleading at best.

Potential issues, based on the online survey, appear to be related to the smoothness of the map interaction, the overall complexity of the map-screen scene (which nevertheless is rated as better-than-OK), and the minigame results-screen.

The in-person individual test sessions are more helpful in pinpointing and identifying issues (see factors in ch. 9.4.2). Several of the identified factors match the online-survey indications: the end-game results screen was found to be not sufficiently clear, and possibility of repetitive gameplay is outlined. There are several reports specifically on the smoothness of map panning, and an overall tendency to mix up highscore-points from single game 'run' with total accrued points, as well as indications towards gameplay repetitiveness. These, then, are the primary issues that need to be solved (see Appendix D for application changes implemented based on these results, but not publicly tested due to scheduling/deadline conflicts).

As with online survey results, the overall impression from most individual test participants is positive, with favourable ratings on visual style, animation smoothness, interface ease of use and game difficulty.

The number of individual-session tests was limited by time, availability, and involvement in the project (people involved in game design workshops did not participate as “testers” to avoid result skew). Thus, all in-persona participants have been sampled from Mittuniversitetet's student & staff population (correlat-
ing to the loose target-demographic definition of 14-40+ year-old smartphone users.)

10.1.5 Hardware system functionality
Due to submission deadline scheduling, the prototype on-bus lighting-system could not be deployed on buses in time for submission. Thus, no observations on the passengers' reaction and feedback could be gathered for this report. The prototype system functionally fulfils the feature requirements – GPS location detection, LED control signal switching, communication with AppEngine backend system.

10.2 Goal completion
The majority of goals outlined in section 1.4 have been fulfilled – a mobile entertainment system has been planned, designed and developed, based on existing field knowledge (the decision to develop a mobile game to better motivate user satisfaction, the choice of development platforms, the consideration of gamification aspects in design process).

A client-side application was developed to function in a “live” deployment and fully communicate with the backend; according to performance tests, the client-side application performs without technical issues and conserves potential billable resources (network usage). User-tests indicate that the application is interesting and generally well-received, though full public deployment will show whether that indication holds.

A back-end system was developed and deployed on Google's cloud platform, and appears to function well with the client-side application, executing all requests in sub-halfsecond timeframes in live scenarios.

A lighting-system hardware prototype was designed and built, and fulfils the functional requirements. However, deployment of the prototype could not be executed before the deadline of this thesis-project (at the time of writing, May 2015, the estimated deployment is on the first week(s) of June 2015), due to marketing-campaign coordination necessities for Din Tur. Therefore, passengers' reaction to the lighting system could not be included in this report. This goal can not be considered to be completely fulfilled as stated in introductory section; however, it is not an indispensable part of the project, and is only used for further promotion of the application/game itself, without feeding data back into the overall system.

The application has been user-tested, application and back-end have both been performance-tested, and lighting system has been segmentally feature-tested. The results of these tests have contributed towards this chapter, and further improvements to the application (ref. Appendix D).

10.3 Ethical aspects
There are three ethical contexts in which to consider the developed application. On one hand, does the resulting game provide any benefit to the public? And on
the other hand, is the game exploitative? Furthermore, does the GPS functionality violate user privacy?

One of the overall goals of the project is to entice people to take public transport instead of other transport modes. As public transport is widely known to be less environmentally destructive than personal automobiles, any increase in public transport users is, by implication, a positive outcome. Furthermore, if this project helps improve the public image of local public transport, then transport agencies may have an easier time in securing funding for further operation and expansion from the municipalities. Of course, the flip-side is that public transport is still not as environmentally friendly as bicycles/walking – thus, if the majority of passenger increase comes from the cycling/walking demographic, then the resulting impact might not be as positive.

The exploitativeness is an easier factor to evaluate. The game very deliberately does not include any form of microtransactions, subscription fees or paid-for unlockable levels. These are, in general, the primary means for gamification as a way to profit, and their absence makes the game significantly less exploitative. Moreover, the game’s design is not oriented towards explicit advertising, nor intended to supersede constructive activities – the game is intended and designed towards being a quick, fun way to spend brief commuting times, and towards being a tool for making public transport itself appear more modern.

However, it is possible to turn this game into 'exploitationware', via the addition of such features as item-shops, paid expansions, in-game advertising and personal-data-mining background code. That, though, is a concern outside the scope and possible control of this project.

User privacy is not compromised by the inclusion of leaderboards on GPS-tagged busstops, due to the following factors: - The users are only ever identified by their freely-selected alias; - User GPS data does not leave the end-device; - Leaderboards only display the 4 highest scorers on the busstop, without an associated timestamp, and without highscore decay; - There is no facility for searching for a user or alias (outside of Datastore, at least), and no possibility for locating where the user has played games (unless they are a high-scorer at every busstop, which would be highly unlikely).

10.4 Usefulness

This is intrinsically a development-focused project. Therefore, the contribution to the scientific and academic fields of computer engineering / software development may appear to be limited. Nevertheless, based on the results and final project state, there are several ways this project contributes knowledge beyond the immediate benefits to Din Tur and bus commuters in Västernorrland.

This project can serve, especially after public deployment in summer 2015, as a case example/study for the use of modern technology as means of improving day-to-day routine processes (such as commuting). By examining the public response, by reaching out to Din Tur to obtain aggregated statistics on the influence of this project, it may be possible to draw larger conclusions towards the
modern public's receptiveness towards “smart technology” and the use of entertainment software as a promotional product. At the very least, this shall serve as a modern, non-standard example of human & smart-technology interaction outside of old peoples' nursing homes (the focus of a majority of research on the subject).

Likewise, this project (potentially in conjunction with post-project public response information) can be used as a study in non-exploitative micro-game design, and the integration of games (not gamification) into everyday procedures, as well as an investigation in effectiveness of using a game as a promotional tool, rather than as platform for advertisement-based promotions.

For corporate interests, this project may indicate the feasibility of using game-oriented solutions to benefit their corporate public perception.

For academic interests, this project can serve as the basis for social research into human-computer interaction (to use the generic term...), further studies on PR & public image perception, as reference/inspiration/basis for similar projects or investigative studies focusing on the results of using non-standard products for innovation.

And finally, this project helps promote the perception and use of videogames as academically viable and socially positive, constructive tools, and as legitimate alternative to “regular” software development.

10.5 Future work
Immediate further work on this project would be related to solving the major and minor negative factors outlined in the user-tests, and performing minor application updates. The public release also needs to be conducted shortly after the submission of this report. Likewise, post-public-release, the backend and lighting system may need to be examined in how effective they are towards their respective tasks, and what improvements could be applied.

Further down the line, the general public's reaction to the application/game/project could be evaluated, and on the basis of that evaluation, as well as passenger satisfaction metrics gathered by Din Tur, further development and support of the project could be considered. The basic principle might also be applied to other companies, resulting in new product development – or the current application's adoption – towards their specific requirements.

Additionally, after gathering data on public response on this project, further evaluation on the effectiveness and impact of the developed methods could be conducted. Consequently, this project could serve as a reference basis for further studies in similar contexts.
References


Development of interactive entertainment system for Din Tur buses
Elijs Dima
2015-06-12


56
Development of interactive entertainment system for Din Tur buses

Elijs Dima


Development of interactive entertainment system for Din Tur buses
Elijs Dima

Published 2015-01-29, Retrieved 2015-04-20

[34] Unity, “Documentation | Profiler”, http://docs.unity3d.com/Manual/Profiler.html
Retrieved 2015-04-20

Retrieved 2015-04-20


http://d-nb.info/1020022604/34/#page=39


https://www.ingress.com/
Retrieved 2015-06-12

[40] Andrimon AB, “Turf | info”
http://turfgame.com/info
Retrieved 2015-06-12

Published 2011-05-03, Retrieved 2015-05-05


Accessed 2015-03-23
Development of interactive entertainment system for Din Tur buses
Elijs Dima

https://www.assetstore.unity3d.com/en/#/content/13153
Accessed 2015-03-23

http://unitypatterns.com/resource/objectpool/
Published 2013-09-23, Retrieved 2015-04-20

https://webapp-improved.appspot.com/api/webapp2_extras/appengine/auth/models.html
Retrieved 2015-04-20

[47] L. Young, “Connect Google App Engine”,
http://forum.unity3d.com/threads/unity-and-google-app-engine-backend.74624/#post-1778780
Retrieved 2015-04-08

[48] MediaTek, “MediaTek and Japan Display Unveil Industry’s First Mobile 120Hz Display Technology”,
Retrieved 2014-07-15
This document was distributed, in .pdf form, to invite people to participate in the public test process. Participants were also given a link to an online feedback form (see section below).
Public test feedback form

The form was hosted on Google Forms, and made available throughout 05-2015. English version shown; A Swedish-version of the form was made available to some participants, with the same question and answer format/content.

Figure 66: Feedback form page 1
Figure 67: Feedback form page 2
Figure 68: Feedback form page 3
Figure 69: Feedback form page 4
Figure 70: Feedback form page 5
Figure 71: Feedback form page 6
Figure 72: Feedback form page 7

Figure 73: Feedback form page 8
Figure 74: Feedback form page 9

Figure 75: Feedback form page 10
In-person test feedback form

The form mostly follows the public form's tasks, but is designed to be filled out by the tester, during observation of participants' actions and after the test, by interviewing the participant. The full form is 8 unformatted pages long, so the questions are given here in text form instead.

Task 0 / Control:

Instruct user to download and install the game/app. Result? (Success / Success, with assistance / Did not install / Installed, does not launch / Other[____]).

Task 1 / Start:

Instruct user to Start the game, To Register, and to Pick a team (without details on how to do that). Result? (Started successfully / Started, with problems / Did not start / Other:[____]). Observations from side:[______]. Users' comments afterwards on this section of the app:[______].

Task 2 / Playing a game:

Instruct the user to start the minigame and play it. (Do not specify required number of plays, do not imply that they need to play multiple times). Result? (Success / Success with problems / Minigame crashed / Game locked up / Other:[____]). Observations from side:[______]. User's feedback on difficulty: [____]. User's feedback on visual style:[____]. User's feedback on performance:[______]. User's feedback on results screen:[______].

Task 3 / Setting a record on busstop:

Instruct the user to set a highscore-record on the local busstop. Do not tell them where to view highscores, or how they are calculated (without them asking explicitly). How many points did they reach?:[___]. How many tries did it take?:[____]. (After several game sessions) User's feedback on game difficulty & length:[______]. Is the game interesting/challenging?:[____]. Observations from side:[______].

Task 4 / Unlock 5 achievements:

Instruct the user to unlock a total of 5 achievements. Do not tell them how to unlock them, or which menu shows them (without them explicitly asking). Can they find the achievements menu?:[____]. Can they figure out how to unlock achievements?:[____]. Observations:[______].

Task 5 / Reset player stats:

Instruct the user to reset their statistics/progress. Can they find out how to do that?:[____]. Observations:[____]. Feedback:[______].
Development of interactive entertainment system for Din Tur buses
Elijs Dima 2015-06-12

Task 6 / Overall feedback:

After the test tasks are completed, ask the user (test subject) to give an overall feedback on the game. Feedback categories: Visual Style:[___]. Performance: [____]. Gameplay:[____]. Ease-of-use:[____]. How interesting is the game in general?:[____]. Any technical issues?:[____]. Anything they would like to see changed?:[____]. Other feedback?:[____].
Appendix B: Application Network Test

Use case definition & data gathering method

The networking test aims to represent a “regular” app use case scenario: A person getting on a bus, launching the application, registering/logging in, viewing statistics for a couple of nearest bus stops, playing a couple of games, panning the map.

Because online-maps are downloaded only when using the application outside of city limits (Ånga, Härnosand, Kramfors, Örnsköldsvik, Sollefteå, Sundsvall, Timrå, Umeå), two sets of tests are conducted, with user GPS-location artificially fixed within Sundsvall for 1st test set, and in Kvisleby for 2nd set.

The test was conducted against the deployment server system in App Engine, to get accurate response parameters.

Every web-request calling section of application code was wrapped with evaluative debug statements, using Unity's WWW.downloadedBytes parameter for size estimation. The http responses were additionally checked via Chrome DevTools to get response header size (20 Bytes, added to data in accordance with request counts).

Single test run procedure

For each run, the following actions were conducted to generate appropriate web requests from the application:

- Launch application
- Register new user account, pick a team
- On map (once loaded), click on 10 other bus stop icons in vicinity
- Play 3 mini-game sessions
- Pan the map so as to load two adjacent map tiles
- Clear player statistics
- Log in & pick a team again
- Quit game.
Development of interactive entertainment system for Din Tur buses
Elijs Dima

2015-06-12

Numeric recorded data

Used as source for graphs in section 9.1.
Figure 76: Application Network request measurement results. RQ size in Bytes, sum over all requests of that type per run. Average RQ size = RQsize/RQcount. Header size (20 Bytes) already included. Due to homogenous nature, small number of repeat-runs.
Appendix C: Application Performance Test

Measurement Terms

While Unity provides detailed performance debugging output, it is a short-termed view and overly in-depth. For performance in perceivable terms, the FPS metric is often used to estimate game performance. For context, movies in cinema have a framerate of 24 frames per second, television operates at 30 fps, and most computer monitors refresh at 60Hz (providing a maximum of 60 frames per second to the user). Smartphones are mostly limited to 60Hz displays as well – higher-frequency displays are a technical innovation that hasn't yet seen widespread adoption in consumer devices (ref. [48]). Therefore, it is reasonable to set 60 frames per second as an “ideal” target framerate, and estimate 24-30 FPS as acceptable (if low) lower limit due to people's acceptance of cinema/television framerates.

Measurement Method

The game's framerate was measured on a late 'test' version of the game (full functionality) on an Android phone, with a special script logging frame performance data and periodically storing to a separate file on phone's filesystem (to allow subsequent analysis/visualization of data without artificial test setups).

The test device was a Sony Xperia C5303 (“Xperia SP”), a mid-range Android smartphone released in Spring 2013, running Android 4.4.3, with dual-core 1.7GHz processor and 1GB RAM.

During the test, a common activity scenario was performed:

- Launching the application
- Logging in
- Swiping the map, opening/closing UI panels in rapid succession, scrolling scrollable-UI elements.
- Playing a minigame
- Swiping the map beyond tile boundaries, initiating a map tile re-load (offline and online)
- Returning to the start-screen

The test took 130 seconds. During this time, 7001 datapoints (frames) were logged.
Measurement Script

As all scripts attached to game objects execute their “Update()” function once per game frame, additional code was added to a persistent storage script, tracking – per every update tick – the current game scene, the current timestamp, and the time taken between this point and the same point in previous frame tick (this permits to track a full frame execution cycle across all scripts & game-engine actions). To minimize tracking code’s influence, the data was logged as values passed to a StringBuilder, which was stored to a text-file on every thousandth frame. Because the game code manipulates the in-engine timescale, the Time.realtimeSinceStartup was used instead of Time.deltaTime for calculating elapsed time.

Script sections responsible for tracking framerate:

```csharp
// void Update(){
//     currentRealTime = Time.realtimeSinceStartup;
//     // how long did the previous frame take? What is the timestamp for it (well, this, but offset by one frame doesn't matter)
//     framerateResult.Append(Time.realtimeSinceStartup); // X axis: timestamp
//     framerateResult.Append(",");
//     framerateResult.Append(1 / (currentRealTime - previousRealTime)); // Y axis: FPS (1 / recorded framespeed (framespeed = time since last frame )
//     framerateResult.Append(",");
//     framerateResult.Append(Application.loadedLevelName); // levelname for control
//     framerateResult.AppendLine();
//     if(framerateOutputSeparator>=1000){ // write periodically to file to avoid writing file every frame
//         Debug.Log("Writing framerate to file!");
//         framerateOutputSeparator = 0;
//         WriteFramerateToFile();
//     } //
//     previousRealTime = currentRealTime;
//     framerateOutputSeparator++; //
// }

// void WriteFramerateToFile(){
//     File.WriteAllText(Application.persistentDataPath + "/framerateNormalized.txt", framerateResult.ToString());
//     Debug.Log("Framerate result written at: "+ Application.persistentDataPath);
// }
```
Appendix D: Post-test game changes

Due to the deadlines, some changes were made to the application after the player-testing phase, but before the submission of this report. This appendix therefore will list the main changes to the application that have been made as a result of the feedback obtained from player-tests. This section will not cover small changes such as grammar-fixes etc.

Minigame changes

Due to some testers' indicated confusion on the minigame mechanics, a more informative help-screen was designed, with explicit finger-tapping iconography.
Likewise, the Game UI was refined to rely more on icons (for time, player-health and player-points), and additional obstacle-objects were added, to provide more mechanical diversity and entice players to tap on multiple locations on screen instead of just one point (additionally, a translation fudge factor was added to vehicle spawning, to slightly vary their vertical position on screen). As a result, the player's reflexes should be more challenged, and basic cognitive decisions are created – e.g. “should I tap off this enemy car, or should I pick up that extra life? Can I risk both?”. As with car obstacles before, the additional objects operate on a gated random respawn timer and random positioning/speed, thus adding more variety on repeated plays.
Lastly, an arrow was added to connect the player's gathered points and the resulting team status-change. Likewise, “team bonus points” are coloured in the player team's colour, to further indicate their origin. This, and the start-screen, were re-coloured to a white base, to better match the other scenes' UI elements. The “Continue” button was fixed to only become active 2 seconds after the results screen appears, to avoid players accidentally skipping past this screen.

Figure 78: Redesigned game-interface

Figure 79: Redesigned minigame result screen. The ghost is there to add more personality to the game.
Map-screen changes
The achievement star icon was replaced with a “trophy” icon, to provide a more clear indication of the icon's meaning and purpose. The “star” shape has been repurposed to indicate “points”, and has been added to the player-info screen, and throughout the minigame interface, to further associate the icon with game-points.

![Figure 80: Map-screen interface changes](image)

Text/Grammar changes
The in-game text is constantly being changed / improved. Any of the screen-shots featured in this report may display misspelled, or structurally awkward swedish phrases (such as “poång” in fig.80). These are all either fixed, or in the process of being fixed, and subject to change between the date of this submission and the final “v1” public release.

Light-system changes
Post-testing and preliminary report submission, the light-system was fully implemented and encased, to be ready for installation on bus.
Figure 81: Light-system functionality test (Wall AC outlet and portable AC-DC transformer)

Figure 82: Internal structure of the light-system's non-RaspberryPi section