Extending the Limits of Weatherlings:

a Ubiquitous, Educational, Multiplayer, Web-Based Game for Mobile and Desktop Platforms

by

Yunus Sasmaz

S.B. Electrical Engineering and Computer Science, M.I.T., 2009

S.B. Management Science, M.I.T., 2010

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May, 2010

Author_____
Department of Electrical Engineering and Computer Science
May 13, 2010

Certified by_____
Eric Klopfer
Associate Professor, Department of Urban Studies and Planning
Thesis Supervisor

Accepted by_____
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Extending the Limits of Weatherlings: a Ubiquitous,
Educational, Multiplayer, Web-Based Game for Mobile and Desktop Platforms
by
Yunus Sasmaz

Submitted to the
Department of Engineering in Electrical Engineering and Computer Science

May 13, 2010

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

UbiqGames is a project that aims to create ubiquitous, educational, multi-player games. Built under UbiqGames framework, Weatherlings is a web-based educational game, which allows players to collaborate in a networked virtual world, using a browser-enabled computer or handheld device. Weatherlings overcomes two of the most important limitations of educational games up to date: immobility and steep learning curves. It requires a small amount of initial time investment and can be played anywhere and anytime. Yet the first version of Weatherlings built in 2008-2009, lacked important features such as real-time integration which meant that the game could be further improved. This paper explains the extensions designed and implemented on top of this initial game in order to develop Weatherlings 2.0. This new version of the game designs, implements, and integrates real-time weather-forecasting games for US and Singapore with new modules helpful to the existing game; makes improvements on the existing features; builds tools to facilitate test results' analysis for teachers and others; and extends the previously defined framework for UbiqGames by adding important design decisions, features and guidelines from our experiences. This paper also discusses, analyzes and interprets the results of two test studies conducted in Singapore.

Thesis Supervisor: Eric Klopfer
Title: Associate Professor, MIT Department of Urban Studies and Planning

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Prof. Eric Klopfer for his mentorship and supervision throughout my research in his team. His encouragement and support has been invaluable to my work over the past year. I am especially grateful to him for giving me the opportunity to conduct a research in an area that combines two of my greatest passions together: technology and education.

I would also like to thank a few people for their invaluable help during the development of Weatherlings 2.0: Josh Sheldon and Judy Perry for their exceptional management over this project and insightful comments, Michael P Medlock-Walton, Sonny Thai and Susanna Schroeder, members of the development team, for their assistance with the design and implementation of Weatherlings 2.0, Matthew Ng for developing the initial version of the game and defining an extensive framework which tremendously helped me in my research, and John Pickle who have provided us with different types of historical weather maps from different locales of the U.S.

I would like to thank the following members of National University of Singapore and Nanyang Technological University: Prof. Vivian Chen, Prof. Henry Duh, Prof. Pei-Wen Tzuo, Wang Wenqiang, Esther Zheng Huilin and Jennifer Isabelle Ong They have had remarkable contributions to this research, especially in the Singapore test studies.

Furthermore, I would like to thank MIT Electrical Engineering and Computer Science Department and Singapore Ministry of Education for their financial support.

Moreover, I would like to thank Burkay Gur for his help with graphics design, Ezgi Hacisuleyman and Burak Tekin for thoroughly reviewing the earlier drafts of this thesis and providing remarkably useful feedback. I am also thankful to Alcin Hakca, Burkay Gur, Mekan Yusupov, Emre Armagan, Ezgi Hacisuleyman and Mustafa Yasin Kara, my close friends who have supported and motivated me during the formation of my thesis. In addition, I would like to thank Mustafa Ali Yorukoglu and Ahmet Kaan Toprak, my friends whom I have had the longest and whom I consider to be my best friends for believing in me in everything I do without question. Besides, I would like to thank all my friends at MIT and in Boston with whom I spent the best 5 years of my life, so far.

And finally, above all, I would like to thank my parents and my brother, Mehmet, Emel and Emre Sasmaz, for their selfless love, never-ending support and confidence in me which is my utmost source of inspiration and motivation.

TABLE OF CONTENTS

1. Introduction 9

2. Motivation 12

3. UbiqGames 13

    3.1. History 14

    3.2. Vision and Principles 15

    3.3. Technology 16

    3.4. Future 16

4. Weatherlings 1.0 18

    4.1. Overview 18

    4.2. How the Game Works 19

        4.2.1. Cards 19

        4.2.2. Battles 19

        4.2.3. Obtaining new cards and leveling up 21

    4.3. Technologies 22

5. First Test Study in Singapore 23

    5.1. Methodology 23

        5.1.1. Sample 23

        5.1.2. Methods 24

    5.2. Findings and Discussion 25

        5.2.1. Gameplay Patterns 26

        5.2.2. Peer-to-Peer Interactions 29

        5.2.3. Motivation of Student Interest in Content Domains 30

        5.2.4. Discussion 30

        5.2.5. Information Extraction from Database 31

6. Weatherlings 2.0 36
6.1. Singapore Challenge 36
6.1.1. Game Play 36
6.1.2. Implementation 48
6.1.3. Integration of Challenge into the Main Game 66
6.2. US Challenge 66
6.2.1. Game Play 66
6.2.2. Implementation 70
6.2.3. Integration of Challenge into the Main Game 79
6.3. Changes to the Pre-Existing Game 79
6.3.1. Database 79
6.3.2. Back-End 80
6.3.3. Front-End 82
7. Second Test Study in Singapore 85
7.1. Methodology 85
7.2. Findings and Discussion 85
7.2.1. Gameplay Patterns 86
7.2.2. Peer-to-Peer Interactions 89
7.2.3. Discussion 90
7.2.4. Information Extraction from Database 92
8. Possible Future Improvements 93
8.1. Singapore Challenge 93
8.2. US Challenge 93
8.3. General 95
9. Framework 96
Appendix 100
References 105

1. INTRODUCTION

Today's youth has an undeniable attraction towards computer games. Many kids who play computer games seem to become attentive learners since they devote large portions of their free time to learning every single detail in a game in order to excel at it. (Martinez, 2006). However, these kids might not be showing similar levels of interest in school-related materials (Martinez, 2006). Therefore one might ask if it is possible to make use of kids' attraction towards computer games in order to teach them school-related knowledge.

Recent research (Prensky, 2001; Gee, 2003; Mitchell & Savill-Smith 2004) has shown that games can be very useful in teaching kids about a wide range of areas in an informal, curiosity-driven fashion. Educational games can provide strong motivation as well as increased enjoyment and engagement (Mitchell & Savill-Smith 2004). Fun and interactive environment of these games can provide seamless learning experiences. A large number of educational games have been designed so far, such as "Oregon Trail", and some of these games became big hits over time. However, there are two common limitations that many of these games share. First limitation is immobility. In other words, these games are mainly designed for desktop platforms so kids can only play them when they have enough free time and desktop-access. This is not very convenient since kids do not have access to desktop computers as easily as they do to mobile phones at schools. Second, the games might have steep learning curves which could be a great drawback for some people.

The question then becomes if it is possible to do better than that; and evidently, it is possible. UbiqGames group has been working on developing a framework for creating ubiquitous games which provide solutions to the limitations mentioned above. The games designed in this framework are casual and can be played anywhere/anytime. Weatherlings, my Masters project, is one of the games that have been designed within this UbiqGames framework. Weatherlings is a web-based educational game which allows players to collaborate in a networked virtual world, using a browser-enabled computer or handheld device. It takes advantage of advancing portable devices technology, real-time weather stations, non-real-time weather history, and growing wireless communication technology. From the educational point of view, its purpose is to teach kids about climate and weather, and how to forecast future weather conditions. To explain in more detail, kids are given hourly weather information for a period of time, either in real-time or virtual-time. Then, they are asked to interpret these data points to

- predict near future weather conditions such as temperature or humidity,
- strategize their game play against their friends.

However, since this was a relatively new game when I joined the project group, there were missing features that would improve the game in many ways once integrated into it. For instance, the game only supported non-real-time weather forecasting, some of the existing features of the game were not fully functional and no extensive user-testing had been conducted. Therefore, as my thesis project, I planned to extend the limits of Weatherlings by

- designing and implementing real-time weather-forecasting game for US and Singapore,

- adding new modules helpful to the existing game and improving on the existing features of the game,
- conducting one test study in Singapore and helping with other test studies,
- analyzing and interpreting test study results, and building tools to facilitate results analysis for teachers and others.

This project is funded through a grant by Singapore Ministry of Education and is a collaboration of Massachusetts Institute of Technology and Nanyang Technological University in Singapore. Singapore is chosen as one of the locations for real-time forecasting game as a result of this collaboration.

Note: I will be unofficially referring to the state of Weatherlings before I joined the project as Weatherlings 1.0. And, I will be unofficially referring to the current state of the game as Weatherlings 2.0.

2. MOTIVATION

According to Nicholas Negroponte, people, aged 0-6, experience active learning where they learn by interacting with the world, and observing the results of their interactions (Negroponte, 2006). However, this curiosity-based two-way interactive learning is replaced at later ages by teaching-based one-way passive learning experience. Negroponte, then, suggests that computers can be effective tools to create interactive learning environments (Negroponte, 2006). Weatherlings aims to create such an environment where kids can learn actively by playing, observing the results of their actions and improving over time.

Another objective of this game is to utilize some of the out-of-class time for learning as students are in school only for short amounts of time every day. Yet Weatherlings' objective is to achieve that without interrupting students' entertainment time and for this purpose, it is developed to be fun, entertaining and educational. Its "casual" aspect promises that it does not require a lot of initial time investment. Researchers aim to overcome some adoption barriers of games by designing them to be ubiquitous (Klopfer, 2008), and Weatherlings possesses this feature as well.

In the next chapter, I will explain what UbiqGames is. Then, I will be focusing on Weatherlings 1.0 and its basic design to give you an idea of how the game is played. However, I will not go into the details of implementation, screen design, user testing and findings as these details are explained by Matthew Ng in his Master of Engineering Thesis, 2009 (Ng, 2009). Finally, I will discuss the extensions and improvements I added to the game to create Weatherlings 2.0.

3. UBIQGAMES

Developed by Professor Eric Klopfer, UbiqGames project aims to create a framework for designing ubiquitous educational games which can be used in teaching in a fun and interactive fashion (Ng, 2009). By definition of ubiquitous, these games can be played anytime and anywhere. For instance, a student should be able to play this game at home on a desktop or on a school bus on a handheld device. The games should also easily fit into schedules and be casual, i.e. they should have simple rules, should not need any long-term commitment and should not require special skills. Moreover, the games need to be educational: they should allow kids to apply classroom concepts to practical cases in the game or they should introduce/develop new concepts which will later be formally covered in class. Finally, since the games are to be played outside class time, they should designed to be interesting enough so that kids will choose to spend some time on them in their free times (Ng, 2009).

At this point, it is necessary to explain UbiqGames in enough detail so that the reader can put Weatherlings 2.0 in its appropriate place in the UbiqGames framework. For this purpose, I will start by explaining the history, in other words, the previous games designed within this framework. Then, I will discuss the project vision and principles, and technologies involved. Finally, I will mention what kinds of games will be developed as a part of UbiqGames framework in the future.

However, I will not give extensive explanations for project history, vision and principles, and technology as these are explained in depth by Matthew Ng in his Master of Engineering Thesis, 2009 (Ng, 2009).

3.1. History

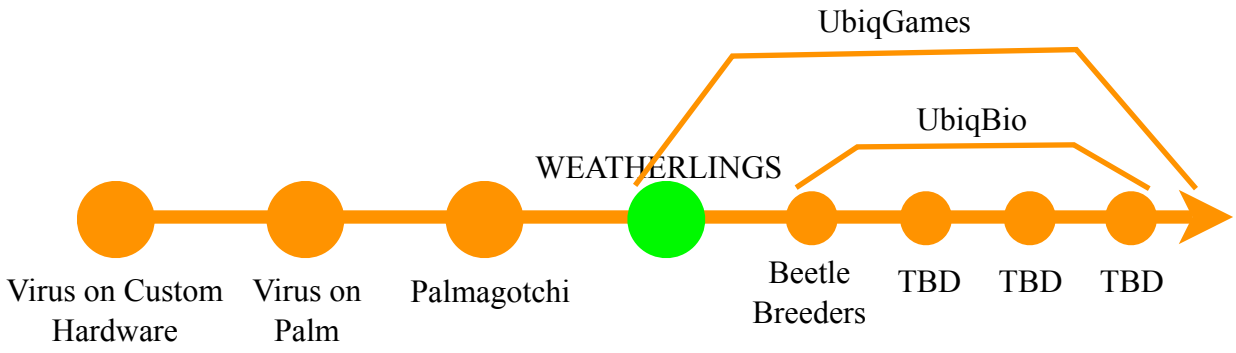


Figure 1. Project History and Future

Participatory Simulations were the first step towards UbiqGames. Participatory Simulations engage people in simulated environments where both technology and social interactions play key roles in reaching a common pre-defined goal. Virus, as an example of Participatory Simulations, was originally developed using custom hardware, and the test studies showed that the game was a success (Colella, 2000). The advantages of the original Virus game was that the data was transparent, and custom hardware prevented any distractions. However, the disadvantages outweighed the advantages: it was costly to build the custom hardware, and it had no re-usability (Colella, 2000).

Later, with growing and cheapening portable device technology, it was possible to use commodity hardware such as Palms to run the games since Palms supported IR peer-to-peer

communication. This highly reduced the cost and increased usability. However, this was not quite ubiquitous as participants had to be present in the same location for the game to be played.

Palmagotchi, “virtual pets with Biology”, was the next game which offered a solution to ubiquity issue: it could be played anytime and anywhere. Users had their own island with birds and flowers, and they were responsible for the evolution on their islands. However, the game was developed for PDAs running Windows Mobile. Although this game was a success for its purposes, it lacked an important feature: platform independency.

Weatherlings 1.0 which came after Palmagotchi is explained in *Chapter 4*.

3.2. Vision and Principles

UbiqGames was started to establish a framework which will allow for creating games that do possess all the good points of the previous games from the previous section, but that do not possess any of their bad points. The visions and principles of UbiqGames are to:

- be widely accessible and widely supported,
- be platform independent,
- be ubiquitous and mobile so that games can be played anywhere and anytime (although internet connection is required),
- be educational and to provide feedback to students and teachers
- support multiplayer engagements which can later lead to classroom discussions,
- allow in-game messaging between users for communication purposes,

- complement to classroom material instead of replacing it.

3.3. Technology

Here are the general technological standards used in UbiqGames:

- Browser-based: This is important for ensuring platform independency. The game is tested thoroughly on Safari, Mobile Safari, and Firefox. At least one of these browsers exists on most platforms.
- Application framework: Ruby on Rails(RoR) was preferred to Google Web Toolkit and jQuery because of three reasons: First, Ruby and Rails are well documented. Second, Rails has been shown to support enterprise web applications. Third, Rails allows easy and quick setup of web applications. (Ng, 2009)
- JavaScript: Ajax allows dynamic content requesting and loading. Therefore, it is used to make the game behave like an application. However, JavaScript is not fully supported on portable devices and this has been a limitation in implementation (Ng, 2009).
- Database: MySQL was preferred because it is scalable, well-supported and easily integrable with RoR (Ng, 2009).

3.4. Future

A new project, UbiqBio, was started within UbiqGames framework in late 2009. This new project aims to promote learning intro level biology in high schools. It will consist of four games covering four areas: Genetics, DNA/RNA/Protein Synthesis, Evolution, Food Webs. These

games will follow the standards set by Weatherlings: it will be anytime, anywhere and optimized for mobile browsers. Also, teachers will be able to keep track of student progress in the games.

The first UbiqBio game, Beetle Breeders, is currently being developed, and it covers classical Mendelian genetics, more specifically, simple dominance, incomplete dominance, co-dominance, sex-linked and polygenic traits.

4. WEATHERLINGS 1.0

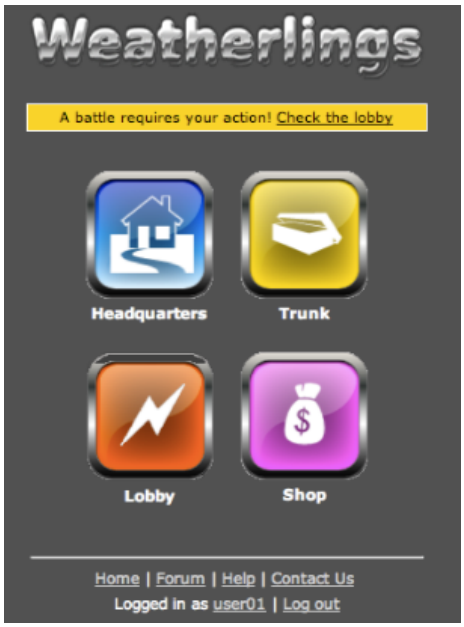


Figure 2. Home page

4.1. Overview

Developed in UbiqGames framework, Weatherlings is a web-based educational game which allows players to collaborate in a networked virtual world, using a browser-enabled computer or handheld device. Players possess some cards and they engage in battles with personalized decks of cards, in which sense, the game is similar to any trading card game, such as “Pokémon TCG” (Ng, 2009).

From the educational perspective, the characters on the cards have powers dependent on the weather conditions. For instance, one card can be very powerful when it is raining and another could be better when it is sunny. Moreover, the arenas in the game are real cities and the weather data presented to the players is real historical data from weather stations.

4.2. How the Game Works

4.2.1. Cards

Players start the game with a set of cards. Each card has the following fields.

- a unique name.

- hit points: the amount of points left before it will be taken off the battle.

- affinity: attraction to a certain weather condition.

- rank: power level.

- moves: set of available moves that it can use.

Figure 3. One of the Weatherling cards

4.2.2. Battles

Players engage in multiplayer battles for the game to proceed. Each of both players picks a deck of cards before the battle and they are not allowed to change the cards on the deck once the battle starts. Then, they choose an arena and number of rounds, and start the battle.

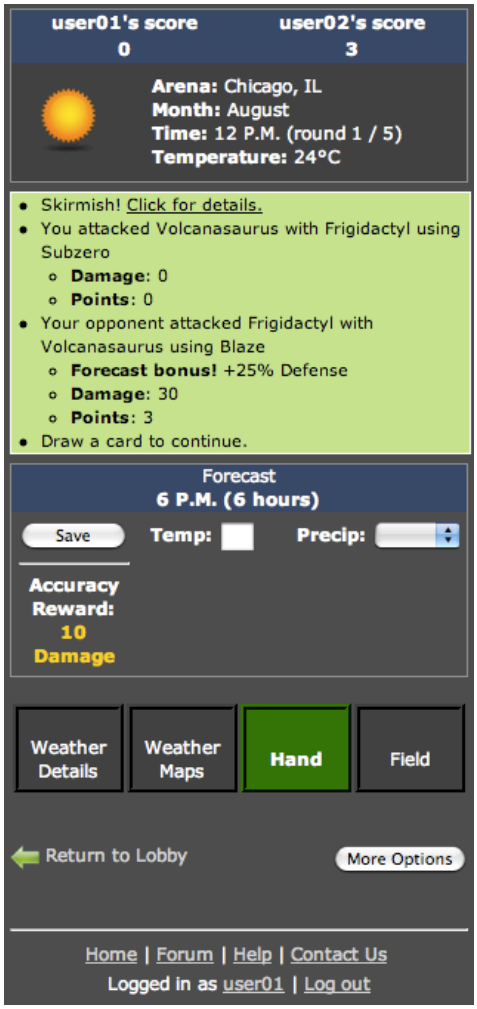


Figure 4. An example battle round screen

A separate piece of the battle is the weather forecasting. At each round, they can choose to forecast what the weather will be at the next round, and they will get bonus points depending on how close their forecast is to the actual temperature.

One important note to mention is that the battle clock is completely unrelated to the real-time. It progresses six hours per round independent of anything. The data for the battle is drawn from the data we collected from weather stations.

4.2.3. Obtaining new cards and leveling up

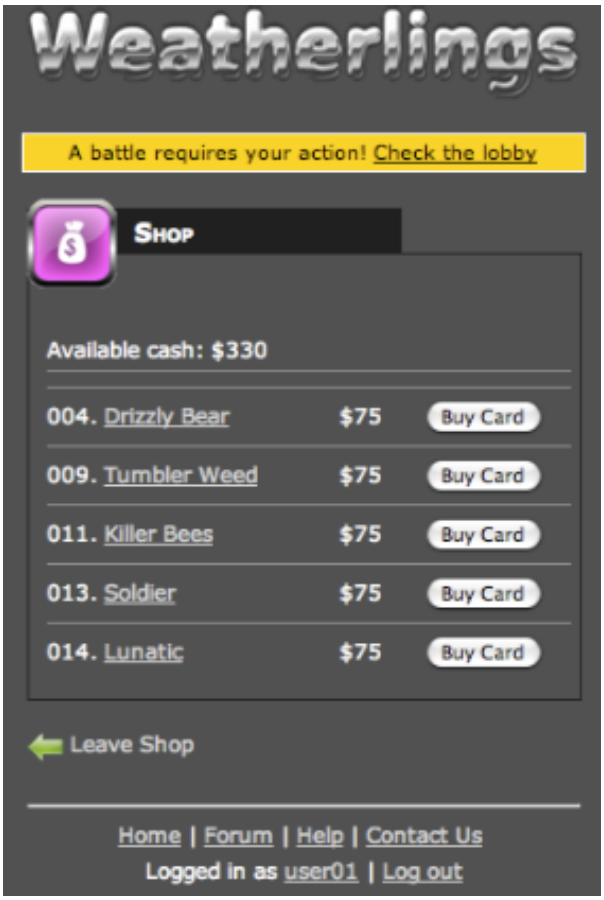


Figure 5. An example list of cards available for purchase by *user01*

At the end of the battle, players get cash and experience points in amounts that depend on whether they win, lose or tie. Using this cash, players can buy new cards. Leveling up occurs when experience points exceed certain threshold values. This is designed to keep the game interesting and challenging for all players.

The cards that are available in the shop are dependent on two factors: the level of the player and the progression of the game, which means that it is also possible to see some new cards in the shop even without leveling up.

4.3. Technologies

Model-View-Controller architectural pattern is used with Ruby (v1.8.6) on Rails (v2.1.0) and MySQL database (v5.0) in creating Weatherlings (Ng, 2009).

5. FIRST TEST STUDY IN SINGAPORE

After Weatherlings 1.0 was completed, Professor Eric Klopfer visited Singapore for an educational game play testing to observe how students would interact with the game and to get a better idea of in which direction to progress with the second version of the game.

5.1. Methodology

5.1.1. Sample (Klopfer, 2010)

The test study was conducted in Pei Hwa Presbyterian Primary School, one of the best schools in Singapore, from September 22nd to September 25th 2009. Twenty students (twelve boys, eight girls), aged 10 or 11, were selected by the teacher from a classroom of forty. Since participation was voluntary and not-graded, the teacher made sure that the selected students wanted to voluntarily participate in the test study (so, in fact, the selection of twenty students was from a pool of voluntary students). Participants were asked to complete a survey prior to test play, and the pre-play survey results indicated that

- All participants (100%) played electronic games.
- All participants (100%) liked playing electronic games. Most participants (85%) showed high levels of interest. The common reasons of liking included “good way to relax and de-stress”, “fun and interesting”, “challenging”, “allows one to do things that he/she cannot do in real life”, “involves a combination of strategy and fighting skills”.
- The most frequently played type of electronic game was computer games (60%). PSP, mobile games and X-Box made up for 25% of the rest.

- 40% of participants played educational games at least once a week whereas 80% of them played entertainment games at least one a week.
- 45%, 45% and 10% of participants were ‘very interested’, ‘interested’ and ‘neutral’, respectively, about learning more about weather.

5.1.2. Methods (Klopfer, 2010)

On Day 1, participants were given mobile devices (Android based HTC My Touch or Android based G1 devices with Android 1.5 running on them) to be returned on day 4. These phones had 3G data plans, which provided participants with anytime, anywhere access to Weatherlings via the browser as long as they had reception or were in range of accessible wi-fi.

After the distribution of the devices, students were given an overview of the game and the hardware and were asked to complete a pre-play survey (which was already mentioned in *Section 5.1.1. Sample*). The goals and mechanics of the game were introduced with real time demos of the game. Given that Weatherlings 1.0 only took place in US cities, the students were told how and why the weather conditions in US could be drastically different from the weather conditions in Singapore due to differences in geographic location, size and latitude. It was necessary for them to understand this difference since they were probably used to non-varying, consistent weather conditions in Singapore. As an example, temperature and precipitation forecasts of the upcoming week for Boston and Singapore were compared in the class. Finally, participants were given a paper-based weather tutorial which is related to the weather information from Weatherlings.

Apart from the game information itself, students were explained that this test study was not graded in any way and would not affect their grade in the class. If this were graded, students might have been worried about their grades; thus, their performance and interest in the game might have been affected. In addition, participants were asked to play the game as much as they want, whenever they want, and wherever they want, until Day 4. Finally, in order to prevent any possible distraction from their classes, participants were asked not to bring the devices to school before Day 4.

On Day 4, mobile devices were collected back and a post-play survey was completed by the students. In addition, researchers had a discussion with the participants about their experiences and ideas about the game. Participants were asked questions such as "What did you like or dislike about the game?", "Would you want other features?" etc.. (more in *Section 5.2. Findings and Discussion*).

The game server recorded participants' interactions with the game throughout these four days.

5.2. Findings and Discussion

After the test study was completed, database entries corresponding to this study and post-play surveys were the resources for analysis. Jennifer Isabelle Ong, a research assistant from National Institute of Education, has prepared a report analyzing the surveys. I have analyzed the database to extract information about gameplay patterns and peer-to-peer interactions through SQL queries.

Now, I will go into details of our findings and discuss the results. I will first focus on the game play patterns observed, peer-to-peer interactions, and motivation of student interest in content domains. Then, I will talk about the techniques I have used to extract data from the database and convert it to a higher level structure. Finally, I will present conclusions from the test study.

5.2.1. Gameplay Patterns

A substantial amount of data was extracted from the database to reveal gameplay patterns. Table 1 indicates that participants engaged in 194 battles in total, however only 136 of them (70% of all) were completed. 194 created battles corresponded to 1264 rounds which meant an average of 6.5 rounds per battle. 136 completed battles corresponded to 883 rounds which also meant an average of 6.5 rounds per battle. This reveals that the reason for 30% of battles being uncompleted was not that uncompleted battles had more rounds on average, but mainly that participants did not have enough time to go back to these battles to complete all the rounds.

Table 1. Battle and Battle Round Information (Created battles include completed and uncompleted battles)

	Created battles	**Completed battles**
Number of battles	194	136
Number of battle rounds	1264	883
Average number of rounds per battle	6.5	6.5
Average duration of a battle (in hrs)	6.8	5.6
Average duration of a battle round (in min)	63	52

Table 1 also indicates that a completed battle lasted for 5.6 hours on average and a battle round lasted for 52 minutes on average. However, these data contain battles which span over multiple

days. Although most of the battles (76%) were completed within 1 day, other battles which were completed within 2 (23%) and 3 (1%) days heavily affected the averages. To see a more accurate average for round durations, we should look only at the battles which were completed within 1 day. This yields 10.31 minutes per round.

Database analysis also answered one of the major questions we had in mind after the test play ended: “How did students’ interests evolve over the four days? Did they get bored after a short while or did they play more often?” Figure 6 shows the numbers of battles initiated on each of the four days: 48, 76 and 54 on the first three days. This indicates that students maintained their interests in the game over 3 days. On the first day, this number could have been affected by the fact that participants received the devices in the afternoon and spent time exploring the devices in the first place. On the fourth day, there were only 16 battles initiated, which might look bad at the first glance. However, participants usually engaged in battles after school ended, and the mobile devices were collected back from participants before the school ended. In other words, 16 battles on Day 4 is not bad news, in fact it could even be good news because a lot of participants started playing the game early in the morning. Number of predictions per day (Figure 7) has a similar distribution to number of battles initiated per day. This was expected since number of battles and number of predictions have a linear relationship.

Although the most accurate way to check if play time interfered with class time would be to analyze the server logs, battle initiation times should give a good enough representation. Figure 8 shows the number of battles initiated in each hourly period over four days, and we can see that

participants' game play times did not interfere with class times. Only a small percentage of battles are initiated before school ends, and we assume that those battles were initiated during class breaks. The most number of battles (63% of all battles) were initiated between 3pm and 8pm. 16% of battles were initiated after 8 pm, and 11% were initiated before noon.

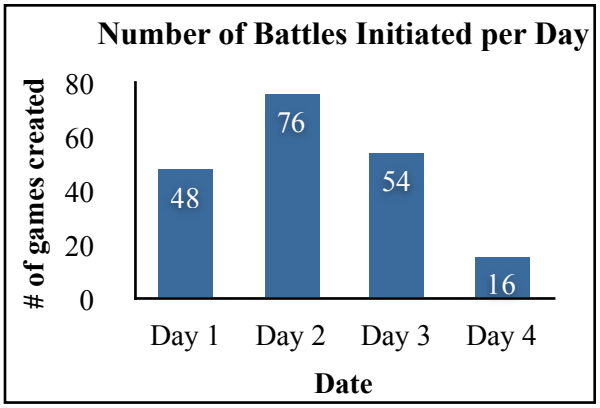


Figure 6

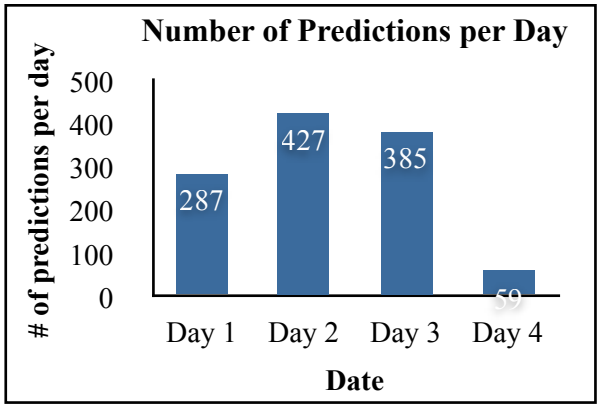


Figure 7

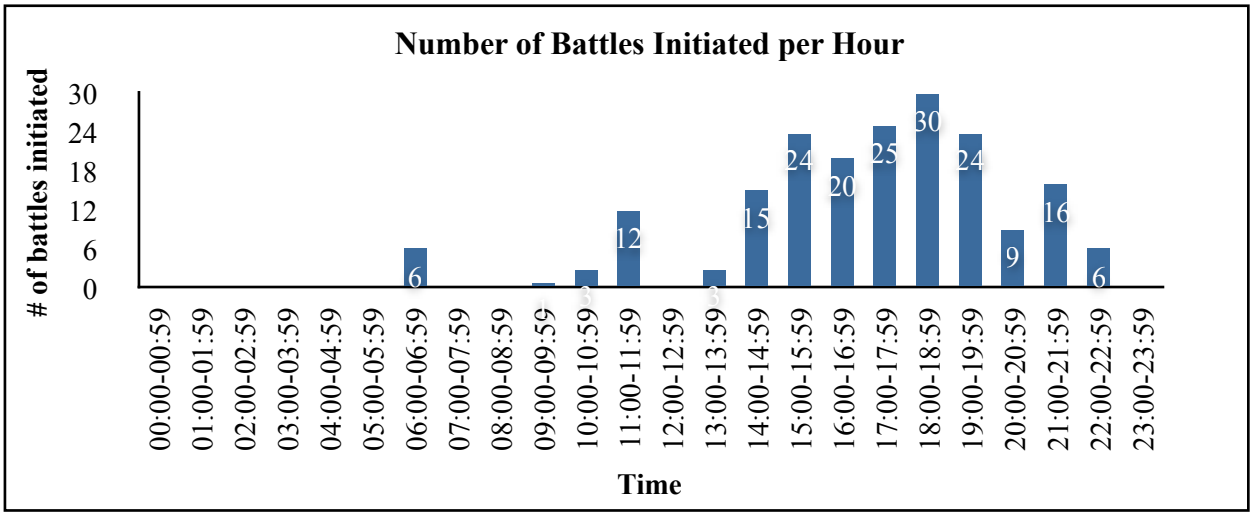


Figure 8

On Day 4, in the discussion with researchers, most participants (75%) reported that they played the game after school. 65% of them reported that they played after dinner. A smaller percentage of them (40%) reported that they played it at or on the way to school. Although database analysis

showed 6 battles initiated before school (Figure 8), none of the participants reported any games before school. Discussion also revealed that 75%, 50% and 35% of participants played Weatherlings in their bedroom, public place in their house and outside, respectively.

In the post-play survey, all participants (100%) noted that the game was fun, and some common responses as to `why` they liked it are: ‘fun yet educational’, ‘challenging’, ‘sustaining their interest’, ‘enjoying the battling’, ‘picking up game skills’, ‘creating decks’ and ‘playing with friends whom they have less opportunity to interact with’. In addition, all of them reported interest in playing the game again.

In short, all of the database analysis, post-play discussion and post-play surveys indicated that participants were strongly engaged in the game, and they wanted to play more.

5.2.2. Peer-to-Peer Interactions

Before the database analysis, we were worried that the participants would only pair up with others who are in their clique. However, further database analysis indicated otherwise, i.e. it showed that the participants played against 9.6 unique players on average (SD=4.02). Every participant played against at least 3 unique opponents (Figure 9) and 60% of the participants played against at least 10 (50% of all participants) unique participants (Figure 9). In addition, on average they played 1.8 battles against one participant (SD=0.70).

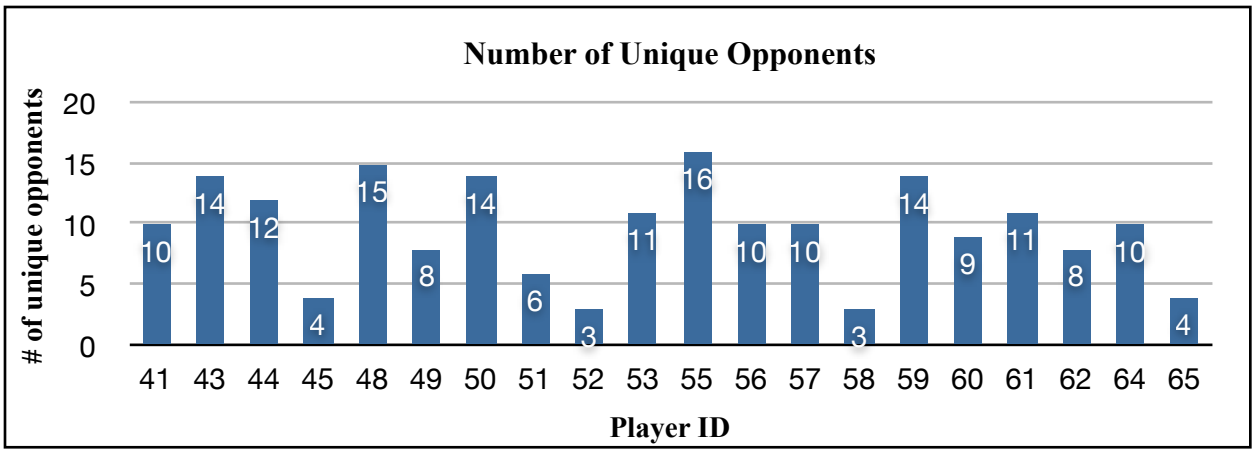


Figure 9

5.2.3. Motivation of Student Interest in Content Domains (Klopfer, 2010)

This test study was successful in fostering interest in learning more about the weather. After playing the game, 90% of participants were very interested or interested in learning more about it. When asked what they would want to know about the weather, 27.3% of participants mentioned ‘learning how to make accurate weather predictions’. Other popular answers were ‘improving skills reading weather maps’, ‘how and why weather changes’ and ‘general/global weather information’.

5.2.4. Discussion

The test study proved to be a success in that the gameplay patterns derived from the database showed that students are highly engaged in the game. This is backed by the survey results which indicate that students were interested in the game and wanted to play more. Moreover, students mostly played the game outside of school, and they played against many different (unique) opponents.

Surveys and server logs reveal that the game was played by the students anywhere and anytime (except school time); therefore, it was truly ubiquitous. It was reported by 95% of students that playing the game taught them more about the weather. In addition, their confidence in weather forecasting improved after the game as the surveys indicate. In the pre-play surveys, only 25% of the students reported that they were ‘very confident’ or ‘confident’ in weather forecasting while this percentage increased up to 65% in the post-play surveys. In short, Weatherlings was successful in being educational as well.

On the other hand, when weather predictions were compared to real weather data, no significant improvement was observed over the four days. This brought the question of whether four days is enough to get better at predictions, or whether students did not try too hard to make good predictions since the weight on the predictions was small compared to the actual game.

One unmeasured goal in this study was the class-complementary feature of this game. However, unfortunately, weather subject was not to be studied anytime soon in this class, therefore it was not possible to observe the success of this feature of the game.

5.2.5. Information Extraction from Database

The database structure behind Weatherlings is shown in Figure 10. SQL queries were used to retrieve data from the database. Most often, the query results were converted to CSV or Excel files. Excel allowed for drawing the necessary graphs and for easy calculation of desired values

such as averages and standard deviations. For some cases when CSV was used, some helper python scripts were written to parse the information and analyze it.

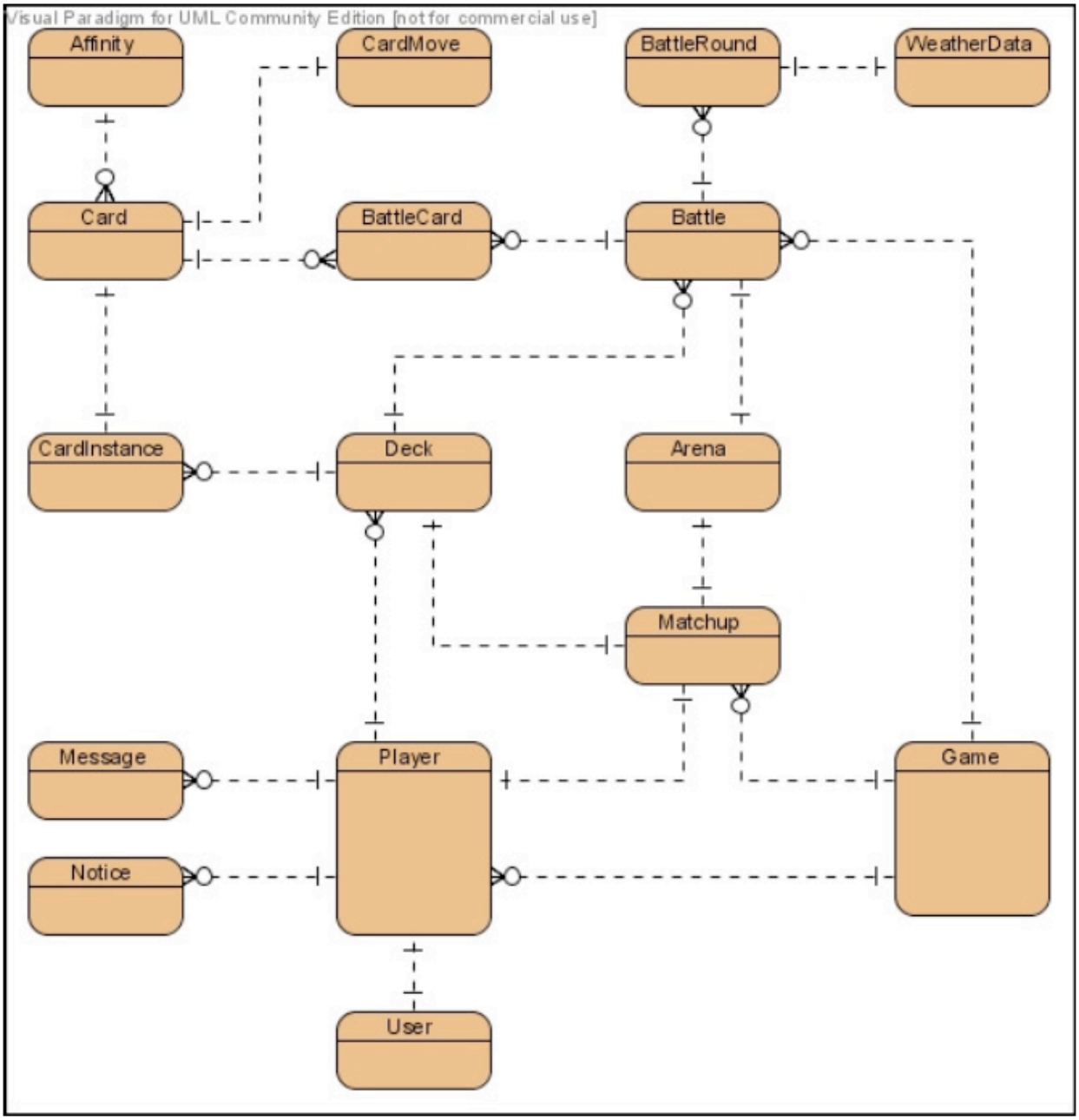


Figure 10. Complete Weatherlings Entity Relationship Diagram
(taken from Ng, 2009)

Next, I will go over some examples of SQL queries used to extract relevant data from database. The most basic query was needed to find the battles played in this test study. The game id of this study was 6, and so the SQL query below was used to extract the battles created in this study.

```
SELECT A1. *
FROM battles A1
WHERE A1.game_id =6
```

To obtain Figure 6, the results of this query were exported to a CSV file and a python script was used to count the number of battles in each day making use of the created_at column of battles. Python was chosen because it allows easy string processing and it was easy to convert the server time (Greenwich Mean Time) to Singapore time (GMT + 08:00). A ruby module would also have been easy to write by passing in the SQL query to database in the module and using the Time library to convert the times.

An iteration of this query gives us the battle rounds which are relevant to this test study:

```
SELECT A2. *
FROM battles A1,
battle_rounds A2
WHERE A1.game_id =6
AND A2.battle_id = A1.id
```

To obtain the number of predictions per day, first the forecasts (predictions) that belonged to these relevant battles are initially needed. It would have been quite easy to do this if the Forecasts table had a field to store a back pointer to which player made the forecast. However, this was one piece missing from it, and forecasts did not have any back pointer which could be

helpful. Meanwhile, battle_rounds keeps track of which battle they belong to, and of the forecast_ids of both players of the battle. Also, the forecasts that were not actual forecasts needed to be eliminated. (This is when the temp field of the forecast is NULL. It means that no temperature was predicted.) These all required a more complex SQL query than the ones above:

```
SELECT A3.id, A2.created_at
FROM battle_rounds A1, battles A2, forecasts A3
WHERE A2.game_id =6
AND A1.battle_id = A2.id
AND (
A1.forecast1_id = A3.id
OR A1.forecast2_id = A3.id
)
AND A3.temp IS NOT NULL
```

Finding the predictions and comparing them to the real data to measure the precision of the predictions also required a complex query. For instance, when we wanted to see how good one single player (player with ID=41 in this case) was in his/her predictions, the SQL query used was:

```
SELECT A1.id, A2.id, A3.id, A1.forecast1_id, A1.forecast2_id,
A2.player1_id, A2.player2_id, A2.score1, A2.score2, A3.temp,
A4.temperature
FROM battle_rounds A1, battles A2, forecasts A3, weather_data A4
WHERE (
A2.player1_id =41
AND A2.id = A1.battle_id
AND A3.id = A1.forecast1_id
AND A4.id = A3.weather_data_id
AND A3.temp IS NOT NULL )
OR (
A2.player2_id =41
AND A2.id = A1.battle_id
AND A3.id = A1.forecast2_id
AND A4.id = A3.weather_data_id
AND A3.temp IS NOT NULL )
```

What this does is first find the battles 41 was involved in, and then the related battle_rounds and forecasts. Next, it finds the corresponding weather data and shows the forecast temperature and real temperature. Rest is handled in Excel.

Discussion of Database Structure

The current database structure, unfortunately, does not store enough information to analyze the results of test studies. Moreover, there are only two time stamps, battle creation time and end time, stored regarding the forecast times, and they are both inaccurate in providing the exact time of the forecasts. This might have led to some issues in Figure 7, since the actual time of the forecast was not known, but estimated.

Keeping back pointers in forecasts and other possible places would fasten data extraction; however, it is not necessary to use back pointers. The reason is that back pointers would require extra effort in the source code, and would unnecessarily crowd the database. However, it is possible to reach any piece of desired information with SQL queries without back pointers. This approach also results in a simplistic database structure which is easy to understand and manipulate.

6. WEATHERLINGS 2.0

Weatherlings 2.0 has many new features to offer to its users. I will group these improvements under three categories: ‘Singapore Challenge’ mini-game, ‘US Challenge’ mini-game and ‘Changes to the Pre-Existing Game’. For each of the mini-games, I will explain the details of game play (user perspective) and implementation (developer perspective). For changes to the pre-existing game, I will discuss the front-end, back-end and database changes as well as the new modules added.

6.1. Singapore Challenge

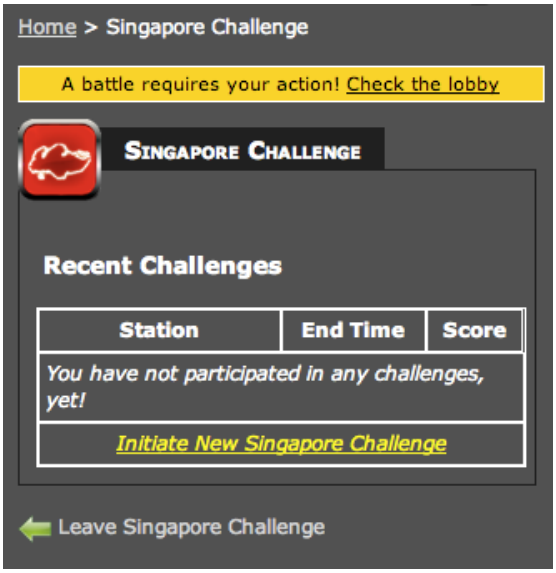
Singapore Challenge adds a real-time dimension to originally non-real-time Weatherlings. Weatherlings 1.0 only used historical weather data which limited the game to only a virtual world. This Singapore Challenge is designed to be a real-time add-on which aims to improve the game by making it feel closer to real world. In this mini-game, Singaporean users initially utilized the given weather history data from a random time in the past to make predictions for the past. However, now they can just observe the current weather conditions in their location and make predictions for future. Nevertheless, Singapore Challenge still provides weather histories for the past seven days.

6.1.1. Game Play

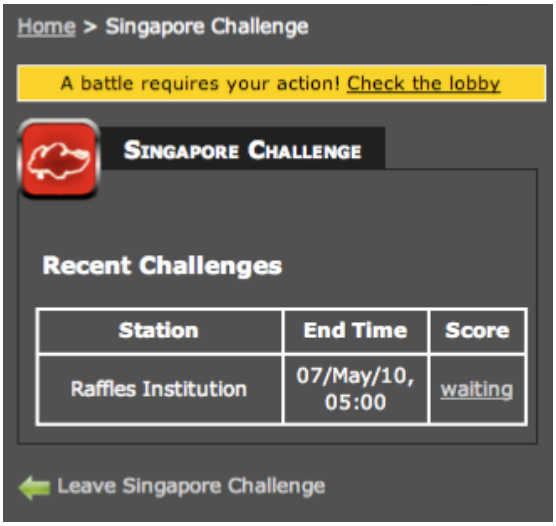
Exploring the Lobby

The first screen users see in the Singapore Challenge is lobby. It displays at most five most finished challenges if available as well as the active, currently on-going challenge, if available.

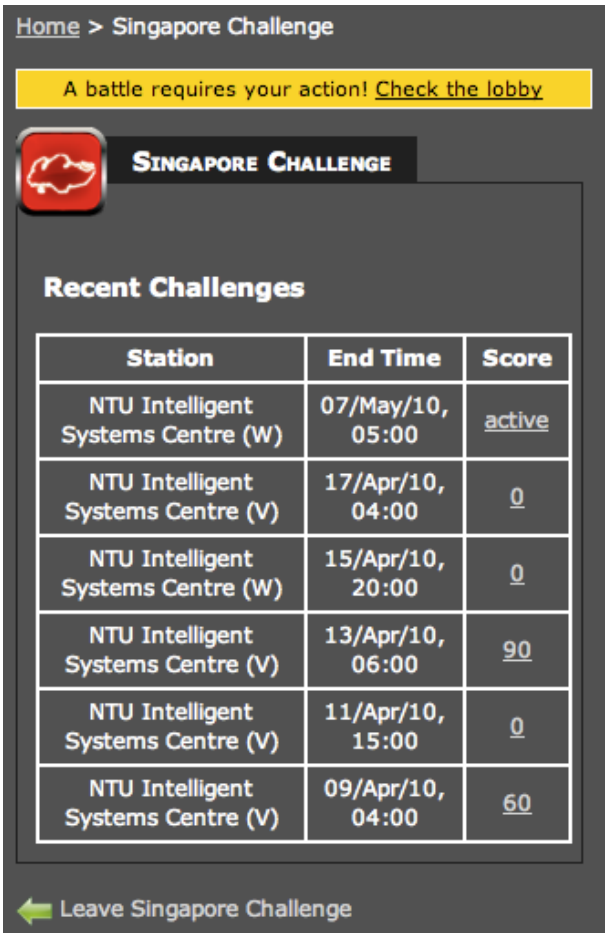
To be simplistic, lobby page only displays three columns of information for each challenge: The station, end time and score(Figure 11c).



a) No challenges, yet.



b) Lobby with an completed challenge.



c) Lobby with many challenges.

Figure 11. Singapore Challenge lobby. Screenshots from different states of lobby.

The first column of information about a challenge is ‘Station’ which indicates whence the weather data is received for the challenge. Second column, ‘End time’ refers to the time for

which the user is supposed to predict. The format is ‘date, time’, ex: 07/May/10, 05:00. The last column is ‘Score’ which displays the points that the user earns in the challenge. However, if the challenge is still active, ‘score’ displays ‘active’; and if the challenge is over, but a score is not assigned yet, it displays ‘waiting’. (Scoring is explained later in this section, *Section 6.1.1. Game Play*)

If the user has not participated in any challenges yet, the page on Figure 11a is displayed with the feedback: “You have not participated in any challenges, yet!”. Notice that this feedback text is personalized. In other words, instead of reading “There are no challenges, yet!”, it addresses the user directly. Although this is a small detail, it is an important technique used in game design to further engage the user in the game.

If the user does not have an active battle at the moment, in other words, if all challenges are finished, a link to create a new battle is shown on the first row of the challenge history table. This link which reads “Initiate New Singapore Challenge” is colored in yellow to stand out from the actual challenge history rows (Figure 11a).

For the purposes of our game, there can only be one active challenge at any given time. The point of this challenge is to offer them new ways to earn cash in the game; therefore give them new motivations to play the game. If we were to allow multiple challenges to be played at the same time, users would be able to make several weather predictions for the same future time point. And, with enough number of trials, at least one of the predictions would be correct. This

could have been prevented by adjusting scoring to punish wrong predictions, however we chose the former method because from an educational perspective, rewarding the 'right' is better than punishing the 'wrong'.

Initiating a New Challenge

A new challenge is initiated when "Initiate New Singapore Challenge" link on the lobby screen is clicked. The first step in initiating a challenge is choosing a station: our system assigns a station to the challenge using the algorithm explained in the section '6.1.2. Implementation' under 'Controllers (C) and Supporting Library Modules'.

Figure 12. Station information.

Next, a page with the station information is displayed to the user (Figure 12). This page includes a map of Singapore with the station location marked in red. There are also instructions below the map on how to proceed to the challenge: ‘click on the map’.

Making Forecasts on the Challenge Screen

The challenge screen consists of three parts: forecast info, most recently received weather data and a menu for weather/forecast history(Figure 13).

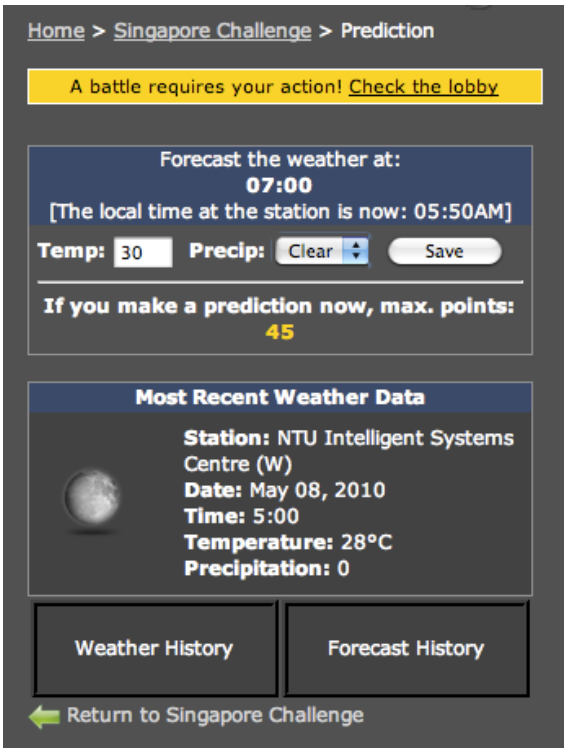
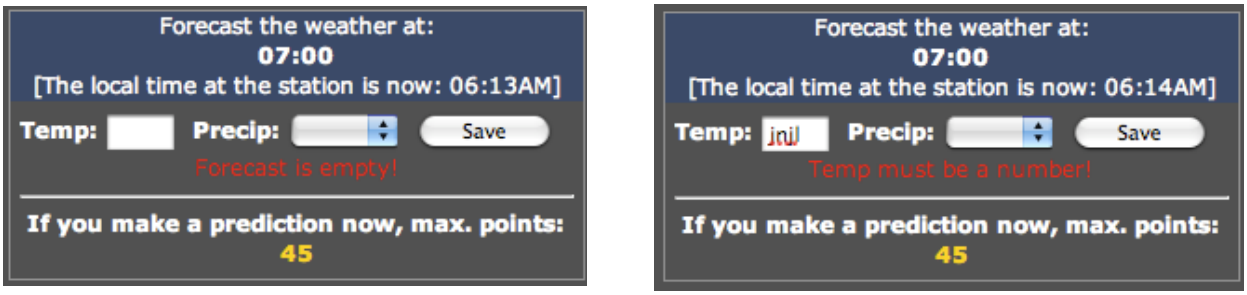


Figure 13. Singapore Challenge screen.

The forecast info box, which is on the top of the screen, is divided into three parts itself: First, it shows the time for which the user is supposed to forecast, and the current local time. Second, it has input fields for temperature and precipitation, with a ‘Save’ button on their right. A drop down menu is chosen for precipitation because there are only three possible states of precipitation (clear, rain and snow) and a drop down menu efficiently serves the purpose here.

However, for temperature, there is wider range of possible outcomes which would make a drop down menu inefficient to use. We decided that a text input field is best here.

The user can save a forecast after filling in the input fields. Forecasts can be saved as long as at least one of the 'temp' or 'precip' fields is non-empty. However, the user is not allowed to save empty forecasts, and if the user tries, the alert in Figure 14a is displayed. Similarly, the alert in Figure 14b is displayed if the user tries to save a forecast with a non-numerical temperature. If there was a forecast placed earlier in the challenge, that forecast data is shown by default in the input fields. This can be considered as a reminder to the users that they made a forecast before because any new forecast will make the previous ones void. Only the last forecast is considered in scoring.



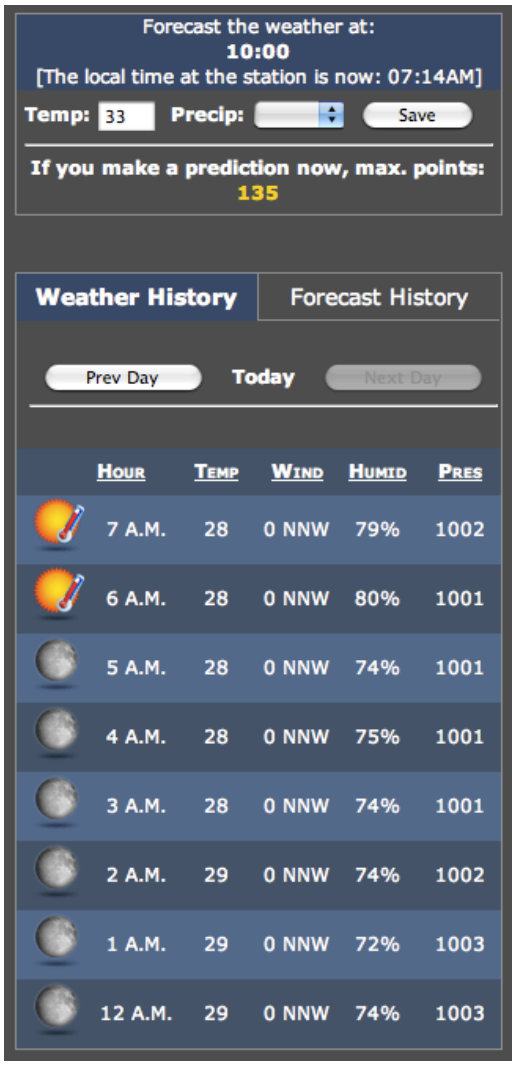
a) 'Forecast is empty' alert.

b) 'Temp must be a number' alert.

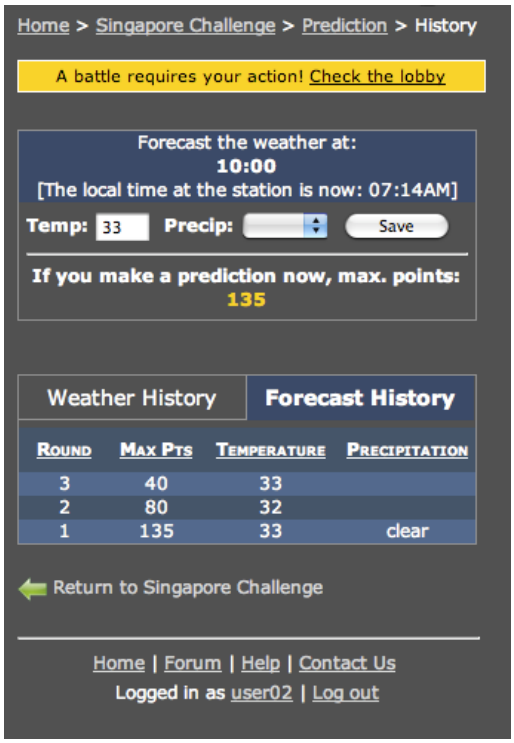
Figure 14. Various forecast alerts.

The final piece of information the forecast info box gives is how many points the user would receive if he/she made a perfect forecast at that time. In this case, 'Perfect' means exact temperature and precipitation.

Most recently received weather data box is quite straight-forward in what it contains. It shows the user the following information: station name, date, time, temperature(in °C since Singapore adopted the metric system) and precipitation. In addition, an icon representing the weather conditions is displayed to reduce the cognitive amount of work required by the user to interpret the time, temperature and precipitation.



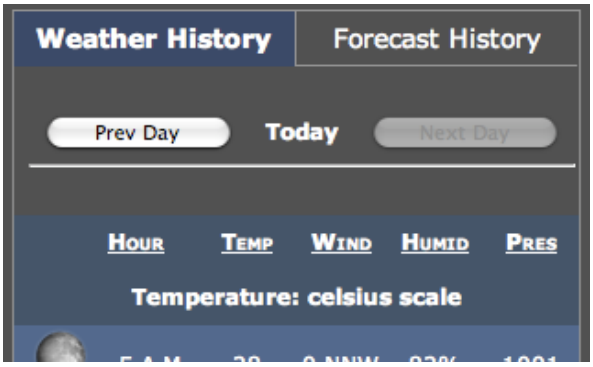
a) Weather History



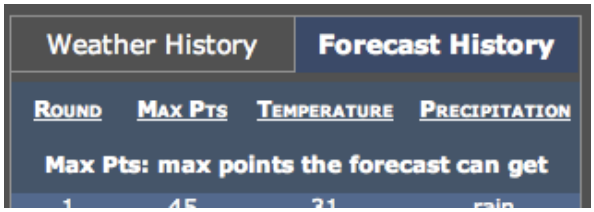
b) Forecast History

Figure 15. Weather and Forecast History

Finally, there is a menu on the bottom of the screen which gives links to weather history and forecast history for the challenge. Weather History (Figure 15a) lists all the hourly weather data from the current day by default. The user can navigate to any day in the last seven days by using the ‘prev day’ and ‘next day’ buttons. Each hourly history row shows a weather icon representing the time, temperature and precipitation. ‘Hour’, ‘temperature’, ‘wind’, ‘humidity’ and ‘pressure’ information is displayed next to the weather icon. Forecast History (Figure 15b) summarizes all the forecasts the user made in this challenge. For each forecast, it reports the round number the forecast was made, max points the forecast can get, temperature and the precipitation of the forecast. Clicking on one of the column headers in weather or forecast history displays explanations for that column (Figure 16a,b). The calculation of max points is explained in the ‘Calculating the Score’ section below.



a) Weather history



b) Forecast History

Figure 16. History column header descriptions.

Since all the history related actions are implemented using Ajax, only the related parts of the page refreshes. For instance, for easy navigation between forecast and weather history, these pages are tabbed and when you click on one tab, only the tab content refreshes which gives it an

application-like feel. In addition, this history page contains the forecast box from the challenge screen on top of the page. This is to allow easy forecasting: once the users are viewing the history, it is likely that they will make a forecast afterwards, and with this design, they do not have to go back to the challenge screen to do so.

Challenge Rounds

Rounds are implicitly integrated into the challenge. The number of rounds for a challenge is determined by the level of the user. The more advanced the user is, the more rounds he gets in the Singapore Challenge. Rounds represent (more or less) the number of hours until the time for which the user is supposed to forecast. However, it is not exactly the hours, rounds are shifted by 20 minutes. The steps behind the formation of rounds are as follows:

1) Number of rounds is the level of the user (let's say, the level is X, so the number of rounds is X),
2) The time for which the user is supposed to forecast (challenge forecast time) is determined by the current local time. If the current local time is
    a. before 20 minutes past the hour (let's say Y:14), then the challenge forecast time is Y+X:00. First round ends and next round starts at Y:20.
    b. after 20 minutes past the hour (let's say Y:22), then the challenge forecast time is Y+X+1:00. First round ends and next round starts at Y+1:20.

To demonstrate with an example, let's assume that the user is level 2, and initiates a challenge at 05:26am. Then, there will be 2 rounds, the challenge forecast time will be set to 08:00am, the first round will end at 06:20am, and the second round will end at 07:20am.

The delay was chosen to be 20 minutes because it was a good balance between giving the user enough time to forecast after viewing the data and having a good amount time to forecast ahead (40 minutes at least). This constant is stored in the constants file in the code and can easily be modified if desired. On an additional note, the challenge page refreshes every minute, so the round switches, i.e. switching from round Z to Z+1, are detected quickly even if the challenge page is kept open for a long time.

How Weather Data is Received

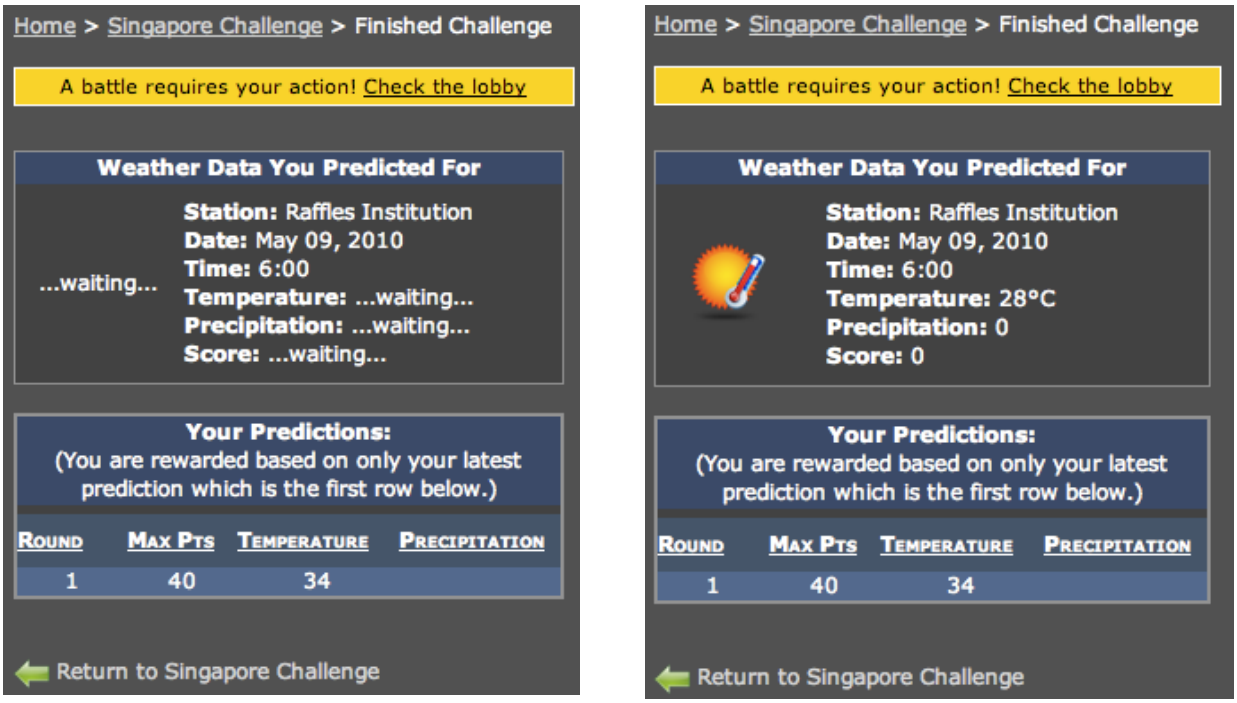
Weather data is received hourly for all of the Singapore weather stations in our database. The data is accessed from a web service by NTU's Intelligence Systems Lab(Intellisys). Data is available in our database usually after 5 minutes the hour.

Viewing Finished Battles

If a challenge is active, its score column will read 'active' and there will be a link to the challenge (Figure 11c). Otherwise, its score column will display either 'waiting' or the points the user earned in the challenge (Figure 11b,c).

Once the challenge is finished, there is a period of time that is spent waiting to receive the weather data corresponding to challenge forecast time, which takes ~45 minutes. The last round of challenge ends 40 minutes before the challenge forecast time (as explained in 'Challenge Rounds' section), and there is a ~5 minute delay after the hour until the weather data is retrieved

to our database. During this period of time, the score column of the challenge data in the lobby reads ‘waiting’, which is also a link, as shown in Figure 11b. The link points to the challenge summary; however, this summary is not complete until the weather data corresponding to the challenge weather time is received (Figure 17a). The score column in the lobby will display the score after it is assigned, and the summary linked to it will be more informative with the actual weather data (Figure 17b).



a) Summary of a challenge which is waiting to receive the related weather data

b) Summary of a challenge which has received the related weather data

Figure 17. Challenge Summaries

The summary includes the predictions that the user saved throughout the challenge, and also displays their ‘round’, ‘max points’, ‘temperature, and ‘precipitation’ information. The weather data box is similar to the weather info box from the challenge screen except that it displays the

score and the data corresponding to the challenge weather time instead of the most recent weather data.

Calculating the Score

The score is calculated according to the following rubrics:

a) Find the last forecast made in the challenge. The score will be solely based on this forecast, and other forecasts will be ignored.

b) Compare the forecasted temperature and actual temperature. If the forecast is far by 0, 1 or 2 °C, the score is 40, 25 or 10. Otherwise, score is 0.

c) Compare the forecasted precipitation and actual precipitation. If the forecast is correct, add 5 to the score from part b. Otherwise, do nothing.

d) Multiply the score from c by the number of rounds in until the end of the challenge. For example, for a forecast in 1st round of a 3-round challenge, this multiplier would be 3 since there are 3 more rounds to go in the challenge including the current one. The motivation behind this rule is that since it is harder to make forecasts for later points in time, the earlier the forecast, the more points it earns.

e) Score from part d is the final score of the challenge.

Therefore, the users can update their forecasts within the same round at no cost. The users can also make new forecasts in the later rounds at the cost of earning less points. The scoring constants such as 40,25,10 and 5 are chosen because this gives a nice balance to the game. A forecast which is only far by 1 °C gives the user enough points to buy a lowest rank card, which

costs 25 points. If the user is more advanced, he probably will not be interested in low rank cards, so he will try to forecast for times further in the future which is the purpose of the challenge.

On a separate note, the terminology ‘max points’ regarding forecasts was used a few times above. The max points is determined by assuming both the temperature and precipitation forecasts are exactly correct. For instance, if a forecast containing a temperature and precipitation prediction is made in the 1st round of a 3-round challenge, the max points would be (40+5)*3 = 135.

6.1.2. Implementation

Weatherlings 2.0 followed most of the standards set by Weatherlings 1.0: Ruby(v1.8.6) on Rails(v2.3.5, notice this was upgraded from v2.1.0) was the development environment with a MySQL database (v5.0). Like in the first version of the game, MVC architecture was adopted in second version as well. To make the code as modular as possible, different pieces of the system which undertake different responsibilities were separated into library modules.

In this section, I will first explain the details of Model(M) in relation to the database tables. After that, I will move on to Controllers(C) and discuss how the system handles input and responds to it. Then, I will focus on the View(V) in more detail although it was mostly covered in Game Play (Section 6.1.1). Next, I will cover the library modules written/used in the implementation of the

Singapore Challenge. Lastly, I will talk about the Weather Stations in Singapore and their reliability issues.

Model (M)

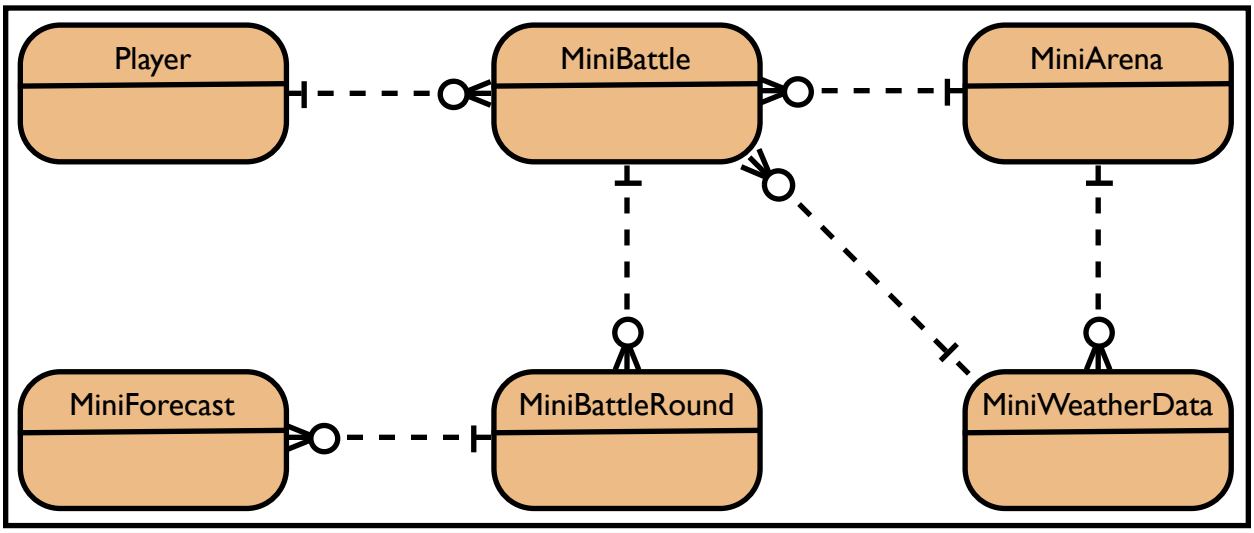


Figure 18. Singapore Challenge Database Models Relationship Diagram

The entity diagram above (Figure 18) shows the relationship between database tables that support Singapore Challenge. 'Player' table is the only pre-existing table and the rest is created from scratch.

Any redundant pointers and columns were removed from the back-end after careful consideration of the trade-off between speed and space. Since SQL queries are very quick to return results, having only single links between tables would not be a bottleneck regarding speed. In fact, the single-link policy helps to keep the database clean and easily manageable. Space, which would be required by the extra pointers and columns, would be close to negligible;

however, it would be harder to manage the database tables from the Ruby on Rails since the developers would have to keep track of multiple pointers instead of one per relation.

Player is the player from the original game and there were no modifications to this class. Since Singapore Challenge is a complementary game to the original game, players can engage in challenges to earn more points. Therefore, players can have multiple challenges, named 'MiniBattle' in the back-end.

The naming inconsistency between the UI and the database exists because we first decided to name the challenge 'Mini Game', therefore everything related to this challenge has the word 'mini' as a prefix in their names. When we decided to change the name to 'Singapore Challenge', it was unnecessary to change the database names as well since the UI and the database are well-separated. Ideally, this inconsistency would not exist, however its existence does not cause any real issues. Here are the correspondences between the UI and the database names:

Table 2. Corresponding UI and Database names of Singapore Challenge

UI Name	Database Name
Singapore Challenge	MiniBattle
Round	MiniBattleRound
Singapore Weather Station	MiniArena
Singapore Weather Data	MiniWeatherData
Forecast	MiniForecast

MiniBattle, Singapore Challenge, is the most complex class as it is directly related to four other classes as shown in Figure 18: Player, MiniArena, MiniWeatherData and MiniBattleRounds. Here are the explanations of these relations:

- Since MiniBattles are single player games, MiniBattle only has one Player and keeps the id of the Player it belongs to.
- MiniBattle takes place in one arena, meaning it receives weather data from only one station, so it keeps track of the MiniArena id.
- MiniBattle has only one MiniWeatherData, the data which corresponds to the time that the player is forecasting for. MiniBattle keeps a column for the id of the MiniWeatherData, however this id is unknown until the related weather data is received, which is about ~45 minutes after the challenge ends. However, although the id of the MiniWeatherData is unknown on the creation of the MiniBattle, the hour and date are known. So, to make this MiniBattle easily searchable in the future when the MiniWeatherData arrives, MiniBattle has MiniWeatherData date and hour columns which are calculated and set during the creation of MiniBattle.
- MiniBattle is divided into multiple rounds which are seamlessly integrated into the flow of the game.

Apart from being in relation to four other classes, MiniBattle has fields to keep track of challenge score, maximum number of rounds, hours per round, status(finished or not), and timestamps (created at and updated at).

MiniArena is one of the isolated, static models. They are created through database migrations reading information from xml files and they are never changed once created. MiniArenas can host multiple MiniBattles and they do have many MiniWeatherDatas, in fact they get one every hour. MiniArenas are stations in Singapore managed by NTU's Intellisys Lab; currently, there are three active stations and many more inactive stations(more on this in *Section 6.1.2. Implementation* under *Weather Stations* heading). MiniArenas store the station code, latitude and longitude information.

MiniWeatherData class stores the id of the MiniArena it belongs to and possesses one of the large tables in our database. For each MiniArena, a new MiniWeatherData object is created and stored in the database every hour, which means about 500 table rows per week given the current number of MiniArenas. Each MiniWeatherData has a date and hour which comes from the observation time, and it is timestamped. MiniWeatherData also holds all the information relating to the physical weather conditions observed by the weather station which are temperature, humidity, wind speed, wind direction, pressure, precipitation, cloud cover and solar radiation.

MiniBattleRound is the class which captures the information about a MiniBattle in hourly chunks. Since each MiniBattleRound belongs to a MiniBattle, it stores the MiniBattle id. In addition, it stores its own round number with respect to the MiniBattle and has timestamps. Initially, each round was planned to have its own scoring but then all the scorings were unified for a challenge, and score is being kept in MiniBattle.

MiniForecast class holds the information from the forecasts of players in the challenge. Since the challenge time is divided into rounds, each forecast is made in a round, and so each MiniForecast belongs to a MiniBattleRound. The fields of a MiniForecast are MiniBattleRound id, temperature and precipitation as well as timestamps.

Notice that all the classes are timestamped. Timestamps have two major advantages for us in this game. First, it helps to make time-sensitive operations; this is important since Singapore Challenge is real time and progression of the challenge highly depends on the time. Second, it helps in the analysis of database records after test studies. Parsing server logs is a painful process; and with timestamps, database tables already contain most of the information we would like to extract from server logs.

Controllers (C) and Supporting Library Modules

Each request is handled by a method and methods are bundled into controllers in Ruby on Rails. When a client request comes through, the response of Ruby on Rails is to look into the config/ routes.rb file in order to determine which controller and which method correspond to the incoming request. Then, the method of interest is called, and appropriate actions are taken. The results of those actions are usually displayed back to the user.

The Singapore Challenge has two controllers: MiniBattleController which mainly handles challenge logic and other challenge related requests, and MiniWeatherController which handles the intermediate steps in the display of MiniWeatherDatas to the screen.

MiniBattleController takes charge once the Singapore challenge lobby is visited. Everything related to Singapore challenge is handled by this controller. This controller frequently makes calls to a helper library module called MiniBattleLogic, which contains all the logic about the challenge itself. To elaborate, MiniBattleLogic helps by encapsulating and modularizing all the logic required to manipulate database models. Specifically, MiniBattleController handles the following tasks.

- Viewing the challenge lobby: This only requires finding all the MiniBattles that belong to the current player and taking only the last five of them.
- Starting a new challenge: This task is immediately directed to MiniBattleLogic where a MiniBattle is created and necessary fields are filled. The player id of the MiniBattle is trivial to find: it is the current player's id. Then, a MiniArena is randomly chosen from the set of online MiniArenas and assigned to the MiniBattle. However, there is no great way to check if a station is online at a given time since we cannot directly send requests to weather stations and receive responses. So, our system assumes that a station is online if there was any weather data received from that station at least once within the last hour. Next, the number of rounds are determined from the level of the player. After that, MiniWeatherData date and time are calculated and set (this will help by making MiniArena search easier once the MiniWeatherData arrives). Lastly, the first round of the MiniBattle is created.
- Showing the current challenge: This is quite easy compared to starting a new challenge. The controller first checks if there is an active MiniBattle. If there is, it finds the last

MiniWeatherData received for the MiniArena of the active MiniBattle and the last forecast made in the active MiniBattle. Then, it passes this information to the View component of the MVC architecture.

- Refreshing the current challenge: This is the same thing as showing the current challenge, so it redirects to the method explained previously. This method is automatically and periodically called by the periodically_call_remote method which is embedded in the current challenge page. In the current design, there is no real need for this method, yet, it can prove useful in the future if the refresh mechanism needs to be replaced.
- Saving a forecast: This method saves the temperature and precipitation values, which the user entered, into a MiniForecast. However, if both temperature and precipitation values are empty or if the temperature is not numerical, the forecast is not saved and user is alerted about the error.
- Canceling a challenge: Currently, this functionality is not utilized. It used to exist in the older versions of the Singapore Challenge, however it was decided not to give the user the freedom to cancel a battle on demand since this feature was not useful.
- Showing summaries of finished challenges: Final feature is to extract information about a finished challenge to pass to the View. Challenge's MiniWeatherData, if received already, is found along with MiniBattleRounds and MiniBattleForecasts. This information is processed in more detail in the View.

MiniBattleLogic includes other functionalities such as updating a MiniBattle, and calculating and updating a MiniBattle score. These functionalities are called by other modules, which will be discussed in 'Other Modules' section below.

The code which handles requests for viewing Weather and Forecast history is placed in **MiniWeatherController**, shown in Figure 15. Since this task is isolated from the rest of the functionalities, it makes sense to separate this part of code from the rest into a new controller.

Views (V)

Views are extensively discussed in Section '6.1.1. Game Play', since Game Play is mostly about the user interacting with the UI. In this section, I will not be duplicating information from section 6.1.1., but I will discuss some design decisions and other important implementation details behind the UI.

As the MVC framework goes, when the controllers end their interaction with the model regarding a request, they send some information to the views to display it to the user. Singapore Challenge's UI followed the UI standards set by the initial game. The screen width is constrained by 300 pixels and height is usually small enough so that it does not require the user to scroll up and down. Considering the internet speed of mobile devices (3G, Edge, etc.), minimal amount of data is transferred between the client and server for each request from the client. Utilization of AJAX helps with the speed by allowing refreshing only certain parts of the page.

One important design decision is about the UI of the challenge lobby page. This page needs to allow for creation and resuming of Singapore Challenges, and has to present a history of previous challenges. To render the design simple, only one table is used. The link to initiate a new Singapore Challenge is put in the first row of the table. This link does not show up if there is an active challenge, since it would be unnecessary. The table was designed to contain only the crucial information about old challenges: station, end time and score. There was no need for a fourth column that links to challenge history or challenge resume, since any of the existing three columns could have been turned into a link. The score column is chosen because the score is shorter than the others and can better convey its affordance to the user. Also, the number of previous challenges to be displayed is limited to five because five is not a small number and five rows fit well to the iPhone screen.

After the user created a challenge, we did not want to directly take the user to challenge screen, but take to an intermediate page before the actual challenge starts. The purpose of this intermediate page would be to give the user information about the station that will be feeding data to the challenge. As a result, the page on Figure 12 is created.

Another important design decision is how to present data to the user in the current challenge screen so that least amount of user's cognitive effort would be necessary to process the data. The information for this page has three major cognitive components that would require the user to focus: forecast, weather data and options to view histories. So, these are divided into different components in the screen. Notice that the 'save' button placement from the initial main game is

switched to come after the temperature and precipitation. This is changed to follow the flow of thought: the user first enters information and then clicks on the ‘save’ button. In addition, AJAX animations for saving empty forecasts are updated to be more responsive.

Also notice that there are breadcrumbs on the top of all the pages and back arrows that link to previous pages. Those give the user a good amount of navigation control on the website.

Other Modules

Other modules were written and some third-party libraries were used to support Singapore Challenge. This section explains these modules and their functionalities.

RtSingaporeWeatherDataRetriever is one of the core modules in the Singapore Challenge as it handles communication between our server and Intellisys’ server. It has four methods: getLatestData, parseLatestData, processSingaporeData and checkEndingMiniBattles.

The first one, getLatestData, is a web client. However, it was not easy to send requests to the web service and receive responses because the Intellisys’ web service uses Simple Object Access Protocol (SOAP). To elaborate, there used to be an actionwebservice gem which came with the default Rails package and which allowed SOAP services for Rails 1.0. However, Rails 2.0 default package does not include this gem because Rails community moved from SOAP to Representational State Transfer (REST) in version 2.0. Although actionwebservice gem can still be installed manually, unfortunately, it was observed after installing that actionwebservice gem

does not include the web service generator. This is a bug by the developers of actionwebservice gem. To fix the issue, datanoise-actionwebservice gem needs to be installed from http://gems.github.com, and the correct gem needs to be placed in the config/environment.rb file with the following line:

config.gem 'datanoise-actionwebservice', :lib => 'actionwebservice', :version => '2.3.2'

Then, an API class needs to be written for the web service which is at the following end point: http://nwsp.ntu.edu.sg:8080/axis/EScienceRTDataService.jws. The only function of the web service that needs to be called is *getLatestData* which takes the station id as input and returns the XML of latest weather data collected from the station as output. The exact web service API regarding this function and its sample output is available in Appendix A.

Once the service response is received from the *getLatestData* method, the output is passed onto parseLatestData, the second method of RtSingaporeWeatherDataRetriever. This method parses the XML output and creates a hash table which contains all the necessary information for a MiniWeatherData to be created. Since XML parsing is a very common task in computing, many tools already exist for XML parsing, so it was not necessary to write a new XML parser from scratch. REXML, which is an XML processor for Ruby and which is distributed under the standard Ruby library, is chosen for this task in this method.

The third method of RtSingaporeWeatherDataRetriever is process_singapore_data, which is responsible for creating MiniWeatherDatas for each station in the database. For each station, it calls the two functions above, getLatestData and parseLatestData. If there is a valid response, if the station is online, and if there is no record of another MiniWeatherData from the same station

in the database which was created earlier in that hour, it creates the received and parsed MiniWeatherData. The reason that there might be another record for the same station in the same hour is revealed in the Weather Stations section below.

After the creation of MiniWeatherDatas, process_singapore_data function does one more thing: it updates the relevant finished MiniBattles. For each new MiniWeatherData, it searches the database to find the MiniBattles with challenge weather times and stations that correspond to this newly created MiniWeatherData. After that, mini_weather_data_id attributes of these MiniBattles are set to their corresponding MiniWeatherData, and their scores are updated now that the forecasts can be compared to the actual weather data.

The fourth method of RtSingaporeWeatherDataRetriever, check_ending_mini_battles, also has a very important task: it goes through all the active MiniBattles in the database and ends them if necessary. If a MiniBattle's page is open when the MiniBattle's end time arrives, the periodic refresh call detects this and ends the battle. However, if a MiniBattle's page is not open by its end time, this method is required to end the MiniBattle. Therefore, this is run hourly, 21 minutes after the hour. Remember that a challenge is supposed to finish 40 minutes before its challenge forecast time, which means 20 minutes after the hour. This method gives a grace period of 1 minute to the MiniBattles.

During the development of RtSingaporeWeatherDataRetriever, one question in mind was how to successfully and efficiently make the hourly calls to process_singapore_data and

check_ending_mini_battles. Initially, a 3rd party gem called *rufus-scheduler* was used. Rufus-scheduler could be initiated in the start-up of the game and it could run the methods of interest at desired times. Although this approach worked well, the fact that rufus-scheduler runs inside Ruby put some burden onto Ruby. In addition, the hourly method calls were naturally interrupted when the Weatherlings server went down. However, we wanted to collect data more reliably even when the server is down. Therefore, we started using the CRON scheduler of the Ubuntu machine which is running our game on it. This way, the computer makes method calls independent of ruby and ruby server. The server logs of these method calls are also stored in the files inside the log/rt folder. These logs are useful for debugging purposes. With all this said, the rufus-scheduler is not completely abandoned: it is still used in the development and debugging environments since it is easier to manipulate this scheduler which is inside the ruby.

CreateMiniArenaData is another major module written for Singapore Challenge. This automates creation of MiniArenas in the database. It is possible to manually create entries in the database, however automation is more efficient and faster. In addition, it allows for storing the station data in an XML file which can be created by anyone. Then, the XML file is parsed and corresponding MiniArenas are created. This time, a 3rd party library, Hpricot, is used for the task of parsing the XML. A snippet of the MiniArena data XML is provided in Appendix C.

In addition to automating MiniArena creation, the module of interest can be easily duplicated and changed to automate creation of other database models as long as the data is stored in XML format.

GameConstants module which existed in the initial game stores the constants of the game. This module is also used to store the constants about the Singapore Challenge such as MINUTES_THRESHOLD_FOR_PREDICTION, since it is not necessary to have another constants module just for the challenge. The **routes.rb** file in the config folder is also modified to include the routes used in the Singapore Challenge.

Weather Stations

We do not receive data directly from the Weather Stations, but we do receive the data from a web service managed by NTU's Intelligent System's Lab for the National Weather Study Project. Intellisys Lab deployed hundreds of mini weather stations in various schools throughout Singapore and a sensor grid is being developed which will automatically gather information from these stations in real time. Intellisys Lab also created a Google Earth tool to display the stations(Figure 19) and information about them(Figure 20). While developing the Singapore Challenge, this tool was mainly used for debugging purposes, such as checking if a station is online or if our server received the correct data.

The only issue with setting the Mini Weather Stations up in various schools in the country is that Intellisys Lab do not have full control over their management. The stations are connected to the computers in the schools and those computers transmit the weather data to the Intellisys server. These weather stations are relatively reliable in isolation, however, if the computer it is connected to is turned off, there is no incoming data from the related weather station although the station is online. Moreover, schools usually turn off their computers at night to save power or at

other times for other reasons, so data transmission is interrupted quite often. Although Intellisys Lab have tried to convince schools not to turn off the computers, they were not very successful since these computers' main purpose is to serve the school. This situation makes the weather stations in the schools unreliable because our game cannot really know when a station data will not be received due to computer power-off. Since the nature of Weatherlings requires the challenge's weather station to be functioning perfectly around one to four hours, the weather stations, which are unreliable, cannot be used in Singapore Challenge. Unfortunately, there are only three reliable stations out of ~50 stations in total. Two of these stations are located on top of the Intellisys Lab and Intellisys Lab has full control over them which explains why they are reliable. The third station is located in Raffles Institute, which agreed not to turn off their computer.

Figure 19. Google Earth Tool Displaying the Locations of Weather Stations

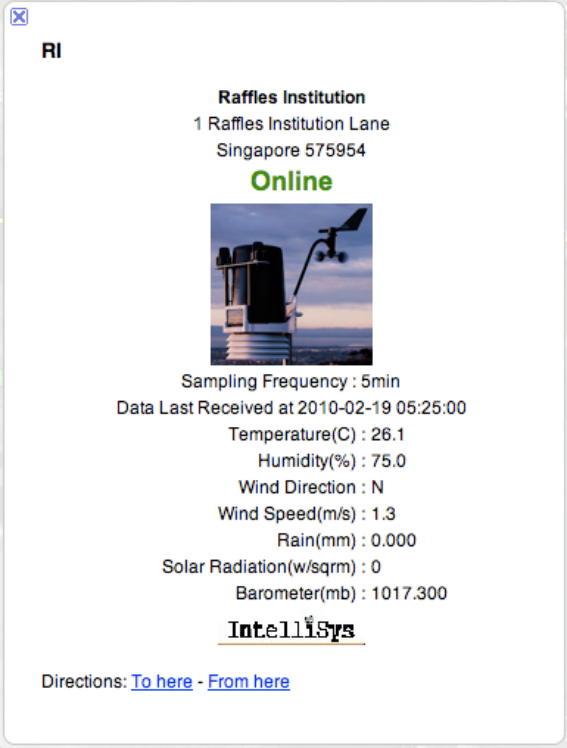


Figure 20. Google Earth Tool Displaying Information about a Weather Station

Intellisys Lab has been looking for other ways of maintaining reliable connections to all of their stations. Currently, they are working on installing Gumstix, small Linux computers (www.gumstix.com), in the schools. These small computers' sole purpose will be to transmit weather data. This way, the schools would not need to turn this computer off for any reason other than energy saving, and since they require small amount of power to run, various schools have agreed to keep these computers on all the time. Once Intellisys Lab achieves a stable project state where many schools are running these small computers, our database will have many more stations to receive data from.

Figure 21. An example of a mini weather station, set up by NTU.

Although some of the stations sample data every minute, some sample only once in 10 minutes. Taking the delays in the transmission into the account as well, data is available in Intellisys' server only after ~10 minutes the hour on average. For this reason, our server runs the script to receive data from Intellisys at 3, 5, 7 and 16 minutes after the hour. The first three times are to receive data from the stations which sample more frequently than average. The forth one is to guarantee that we receive data from the other stations as well.

6.1.3. Integration of Challenge into the Main Game

The integration of the Singapore Challenge into Weatherlings was relatively easy since the challenge is almost completely isolated from the main game. The only shared model is Player. Some attributes of Player such as ‘level’ are needed to be read, and some attributes such as ‘score’ are needed to be overwritten by the challenge. Other than that, the same database and MVC framework are used to create new tables, models, controllers, views and modules.

6.2. US Challenge

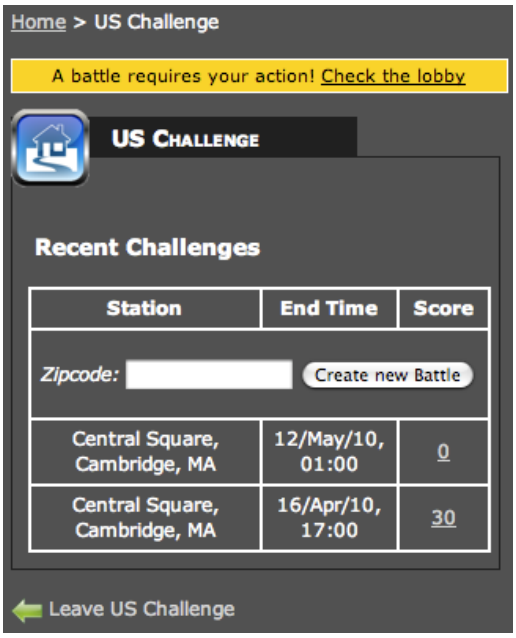
After the Singapore Challenge was successfully implemented, there was still no real-time aspect to the US part of the game. Therefore, a US Challenge was also implemented and integrated into Weatherlings.

The US Challenge is very much like the Singapore Challenge except that it is aimed towards the users in the US. Almost everything about its game play and implementation are identical to those of Singapore Challenge. However, there are a few differences, mostly in the back-end, which are worth mentioning. In this section, I will take the Singapore Challenge as base and only explain the differences between the US and Singapore Challenge.

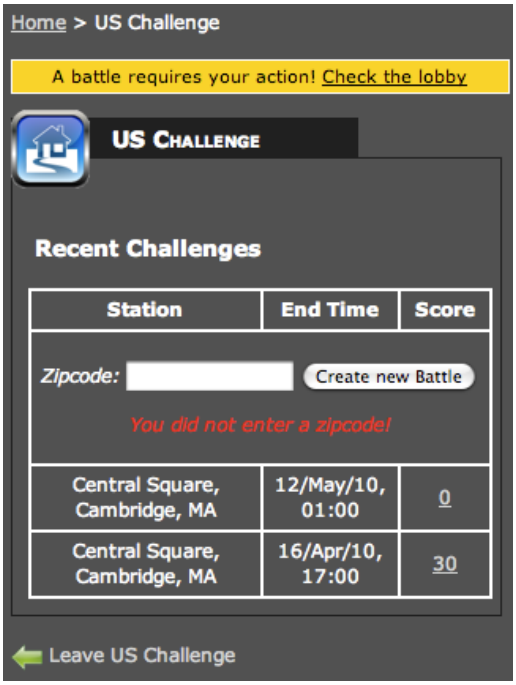
6.2.1. Game Play

US Challenge’s game play is same as Singapore Challenge’s except for three areas: initiating a new challenge, viewing weather history and scoring. This consistency between challenges makes transition from one challenge to another easier for the users who want to play in both challenges.

In addition, the temperatures are shown in °F since the US uses imperial units.



a) Lobby with no active challenge.



b) Alert for empty zipcode.

c) Alert for non-numerical zipcode.

d) Alert for invalid zipcode.

Figure 22. US Challenge lobby (Screenshots from different states of the lobby)

Initiating a New Challenge

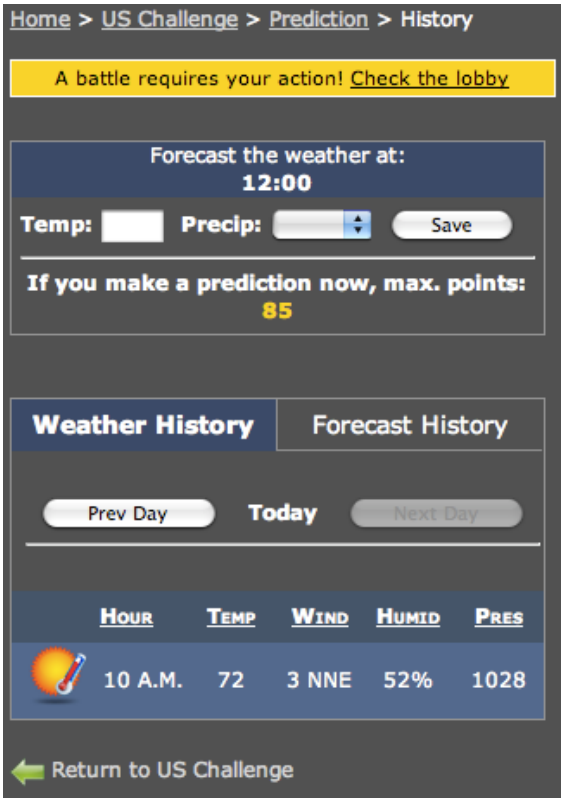
In the Singapore Challenge, users make forecasts for stations which are randomly drawn from the database. This approach works fine for Singapore because it is a small country, stations are at most 30 miles apart from each other and thus will have similar readings. However, as this distance can go up to ~3000 miles in the US, some other mechanism was needed. Eventually, assigning stations according to the zip code was chosen.

As shown on Figure 22a, the 'Initiate new Singapore Challenge' was replaced by a form which takes a zip code as an input and creates the challenge for that zip code. Notice that the users can input any zip code, not necessarily their own zip code. However, it is still expected that most people will want to forecast for their own location, and so input their own zip code. Also, notice the red-colored alerts in Figure 22b,c and d. The zip code input form is not only smart to know if a zip code is entered and if the entered zip code is valid, but it provides animated feedback to the user about the error, usually within one second. The implementation details of initiating a new challenge are covered in the 6.2.2. Implementation section below.

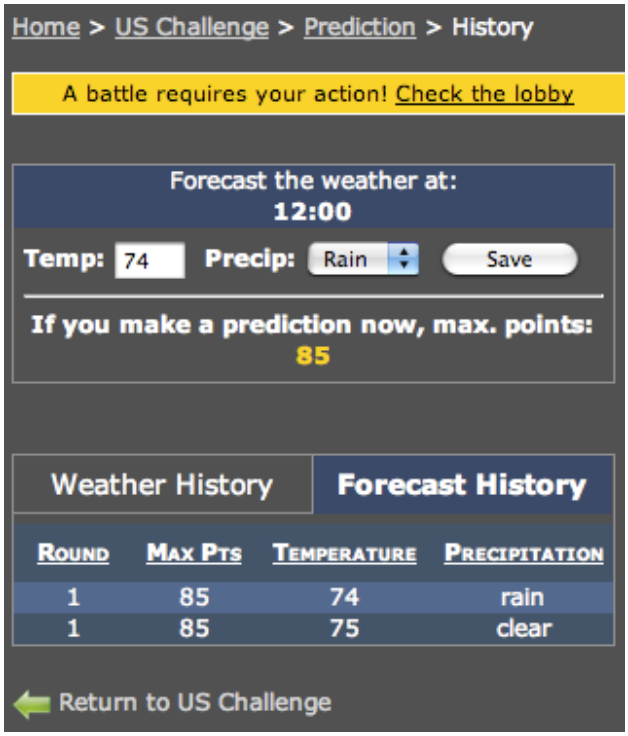
Viewing Weather History

In Singapore Challenge, there are only 3 stations to receive data from, so our server receives hourly data for all 3 stations. However, this approach is inapplicable for the US as our server would need to receive hourly information for each zip code available in the US. Instead, our server only starts collecting data for a zip code when a challenge is created for that zip code.

Unfortunately, a complete weather history cannot be displayed to the user to further educate the user about the weather conditions Figure 23a.



a) Weather History



b) Forecast History

Figure 23. Weather and Forecast History

Calculating the Score

Making accurate forecasts is harder in the US than in Singapore, since weather conditions change more frequently and faster in the US, mostly due to the difference in the latitudes. Therefore, the scoring schema is updated to account for this difficulty in the US forecasts. Here is the new method for calculating the score:

a) Find the last forecast made in the challenge. The score will be solely based on this forecast, and other forecasts will be ignored.
b) Compare the forecasted temperature and actual temperature. If the forecast is far by 0, 1, 2, 3 or 4 °C, the score is 80, 60, 40, 25 or 10. Otherwise, score is 0.
c) Compare the forecasted precipitation and actual precipitation. If the forecast is correct, add 5 to the score from part b. Otherwise, do nothing.
d) Multiply the score from c by the number of rounds in until the end of the challenge. For example, for a forecast in 1st round of a 3-round challenge, this multiplier would be 3 since there are 3 more rounds to go in the challenge including the current one. The idea is that since it is harder to make forecasts for later points in time, the earlier the forecast, the more points it earns.
e) Score from part d is the final score of the challenge.

Max points is also calculated with this method. For instance, if a forecast containing a temperature and precipitation prediction is made in the 1st round of a 3-round challenge, the max points would be (80+5)*3 = 255.

6.2.2. Implementation

Same technologies are used and MVC architecture is followed for US Challenge as was the case for Singapore Challenge.

In this section, I will first explain the changes in the MVC architecture with respect to the Singapore Challenge. Then, I will cover the changes in library modules written/used in the implementation of the US Challenge. Lastly, I will talk about the how we receive weather data for desired zip codes.

Model (M)

The entity diagram (Figure 24) shows the relationship between database tables that support the US Challenge. These models were mostly copied from the Singapore Challenge with new names. Table 3 shows the correspondences between the UI and the database names.

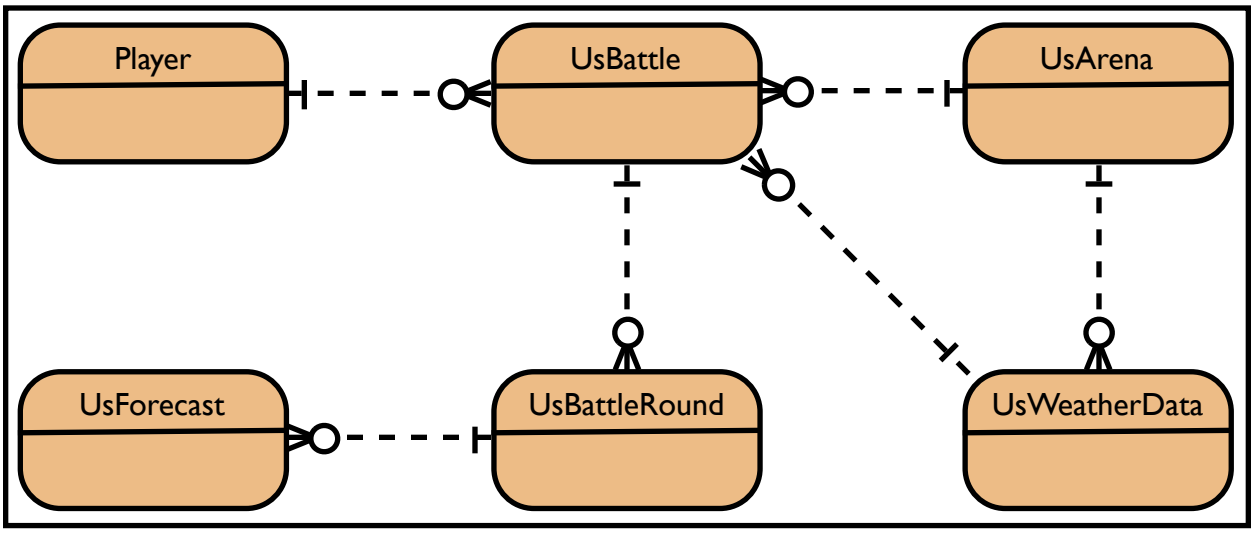


Figure 24. US Challenge Database Models Relationship Diagram

Table 3. Corresponding UI and Database names of US Challenge

UI Name	Database Name
US Challenge	UsBattle
Round	UsBattleRound
US Weather Station	UsArena
US Weather Data	UsWeatherData
Forecast	UsForecast

Besides having all the attributes that MiniBattle has, **UsBattle** has an pws_active attribute (boolean) to keep track of the type of the weather station that feeds it. This is a flag to determine if the station is a personal weather station or an airport station. This is currently defaulted to 'true' since our system receives data only from personal weather stations. However, it is implemented to allow for future enhancement. The reason for using only personal weather stations will be discussed in the Weather Stations section below, and *Possible Future Improvements* chapter will discuss how an improvement could be possible.

UsArenas are also static objects; however, there is no XML file to feed UsArena data to the system since they are created dynamically for the entered zip codes unlike in MiniBattles. Once a UsArena is created, it stays in the database forever. This probably would not be an issue given that there are only ~43000 zip codes available in the US. However, if UsArenas table grows so large that it becomes hard to maintain, the system could be modified to automatically remove old UsArenas from the database. In addition, UsArenas also hold the city, state, zip code and neighborhood information about their stations.

The only addition to **UsWeatherData** other than the attributes from MiniWeatherData is 'observation_time'. This information is later used to determine time zones and appropriately respond to client requests from different time zones.

UsBattleRound and **UsForecast** are structured exactly like MiniBattleRound and MiniForecast. In addition, all the models are timestamped for the same reasons: helping time-sensitive operations and speeding up analysis of a group of database entries, such as the entries from a test study.

Controllers (C) and Supporting Library Modules

The US Challenge has two controllers: UsBattleController which mainly handles challenge logic and other challenge related requests, and UsWeatherController which handles the intermediate steps in the display of UsWeatherDatas to the user. However, before going into the details of these two controllers, I need to explain two helper modules, UsBattleLogic and RtUsWeatherDataRetriever because the controllers are heavily dependent on them. (Notice that the Singapore Challenge counterpart of RtUsWeatherDataRetriever was explained as a part of 'Other Modules' in section 6.1 since it was not called from the controllers).

In the US Challenge, **UsBattleLogic** handles the same functions that MiniBattleLogic handles in the Singapore Challenge. It encapsulates all the logic about the challenge. Specifically, it handles setting up a challenge once an arena is inputted, updating a challenge, ending a challenge, and

calculating/updating the score of a challenge. The details are very similar to those of MiniBattleLogic.

RtUsWeatherDataRetriever's main role is to communicate to Wunderground Data Feed by Weather Underground, Inc (WUI). Specifically, it handles

- getPwsId - Looking up the personal weather station(pws) closest to a zip code: Wunderground Data Feed API provides a GeoLookupXML service to find a city in their database. For instance, the page at http://api.wunderground.com/auto/wui/geo/GeoLookupXML/index.xml?query=94107 provides the information about zip code 94107 in XML format. The information includes the pws's near the zip code and their distance from the zip code. So, it is easy to find the closest station to the zip code. The return value of this function is the 'pws id of the closest pws to the zip code'. If the zip code is invalid, an error is returned. To talk about the logistics, the service output is parsed using Hpricot, the XML parser which was used earlier as well. Appendix D contains a snippet of the GeoLookupXML API.
- createPwsLatestData - Receiving current weather conditions for a pws: Wunderground Data Feed API provides a WXCurrentObXML service to find a city in their database. For instance, the page at http://api.wunderground.com/weatherstation/WXCurrentObXML.asp?ID=KCASANFR70 provides the current weather condition at the pws whose id is KCASANFR70, in XML format. After the relevant information on this page (such as UsArena info) is parsed with Hpricot, it is checked if our database already has a UsWeatherData which corresponds to the parsed UsArena,

UsWeatherData date and hour. If it does, the corresponding UsWeatherData from the database is returned. Otherwise, a new UsWeatherData is created and returned. Appendix E contains a snippet of the WXCurrentObXML API.

- createUsArenaForZipcode - Creating a UsArena for a zip code: First, the pws_id is found using the function getPwsId. If pws_id is not valid, a descriptive error is returned. Otherwise, createPwsLatestData function is called to receive information about the location. Hpricot is used to parse the data, and a UsArena holding the parsed information is created and returned.
- pullCurrentWeatherDataForUsArena - Receiving current weather conditions for a UsArena: The pws_id is parsed from the input UsArena and createPwsLatestData is called with this pws_id. The return value of createPwsLatestData is returned.
- process_us_weather_data - Creating hourly UsWeatherDatas for active UsBattles: First, the UsBattles which still do not have UsWeatherDatas are identified. (Having a UsWeatherData means that the weather data corresponding to the challenge forecast time has already been received and assigned to the UsArena.) Then, the locations (UsArenas) of these UsBattles are determined. For each location, pws_id is found and the latest weather conditions are pulled with the help of the functions described above. If there is no record of another UsWeatherData from the same pws in the database which corresponds to the same date and hour, the new UsWeatherData is created with the pulled data. Finally, all the UsBattles which were waiting for this new UsWeatherData are extracted from the database; their scores are calculated and updated; and their us_weather_data_id attribute is set to the id of this new UsWeatherData.

- check_ending_us_battles - Ending the UsBattles which are over: All the active UsBattles are looped through and they are ended if necessary. This is run hourly, 21 minutes after the hour. Remember that a challenge is supposed to finish 40 minutes before its challenge forecast time, which means 20 minutes after the hour. This method gives a grace period of 1 minute to the MiniBattles.

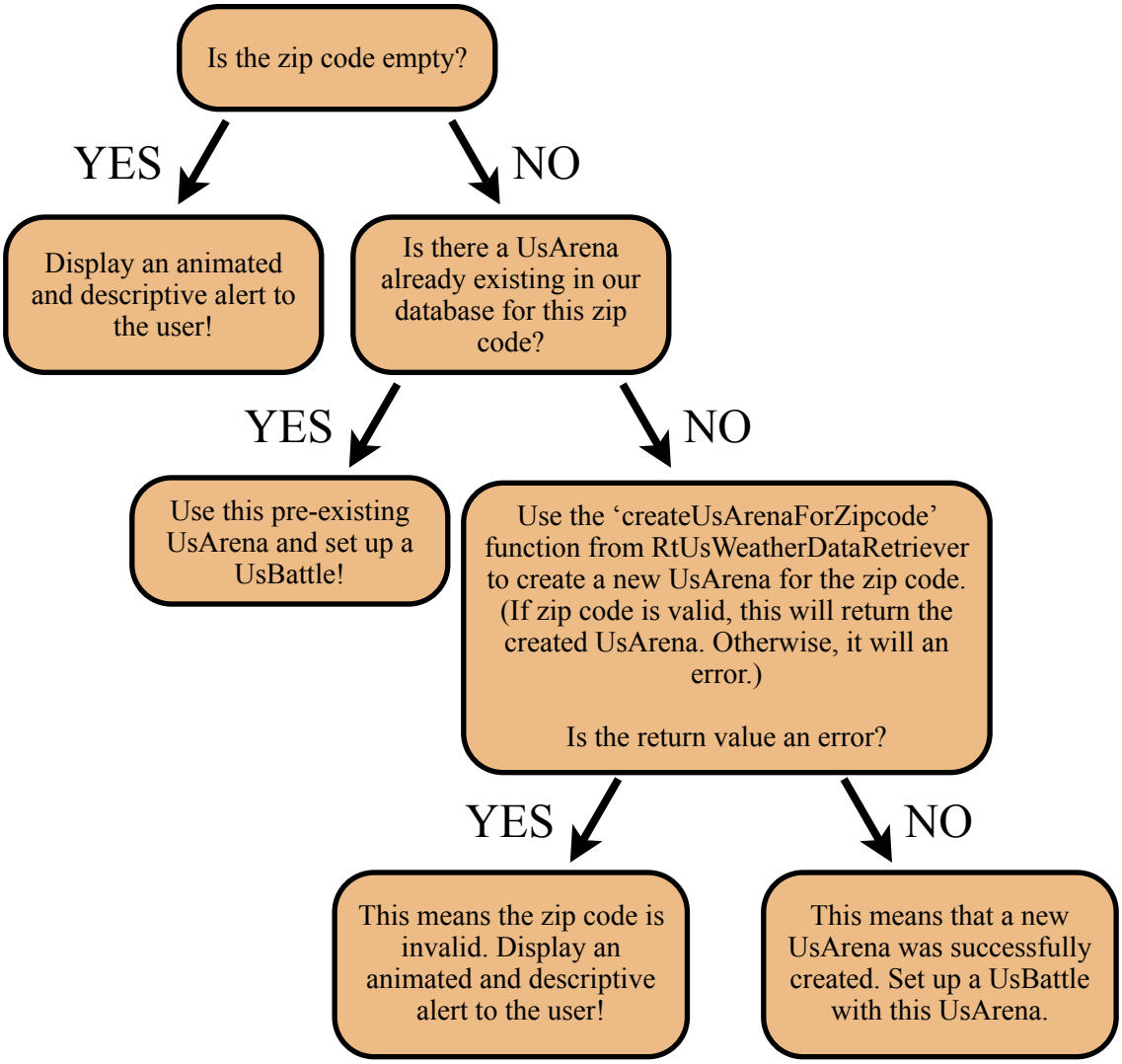


Figure 25. The decision tree for determining the response for a 'Create New Battle' button click in the Lobby.

Now that helper modules are covered, it is time to explain the controllers starting with the UsBattleController. **UsBattleController** behaves in the same way as MiniBattleController in viewing the challenge lobby, showing and refreshing the current challenge, saving a forecast, canceling a challenge and showing the summary of a finished challenge. On the other hand, initiating a new challenge is different in the US Challenge because choosing an arena is now a completely different process. When the 'Create new Battle' button is clicked, the decision tree in Figure 25 is used to determine how to respond to the request.

In addition, UsBattleController accounts for time zones when determining the challenge forecast time. When there is a piece of screen which displays the time (current or future time), it is necessary to display the time in the correct time zone. This is achieved through the observation_time attribute of UsWeatherData saved when the challenge is started. observation_time includes the time zone information, and our system uses it to adjust the necessary times in the challenge. Since Singapore only has one time zone, this is not an issue for the Singapore Challenge.

On a separate note about speed, since our server usually needs to communicate to Wunderground Data Feed before a UsBattle can be initiated, responding to the request is slower in the US Challenge compared to the Singapore Challenge.

Finally, the code which handles requests for viewing Weather and Forecast history is placed in **UsWeatherController**, shown in Figure 23. Since this task is isolated from the rest of the functionalities, it makes sense to separate this part of code from the rest into a new controller.

Views (V)

There are only a few differences in the Views between the US Challenge and the Singapore Challenge, and they are all covered in the "6.2.1. Game Play" section.

Weather Stations

In this challenge, we pull the weather data from the WUI, Wunderground Data Feed of Weather Underground, Inc (we would like to thank WUI for sharing the weather information). This data feed provides services to find weather stations close to a zip code (i.e. looking up cities by zip code in Wunderground city database) and to read real-time observations from weather stations. The API and the urls for the mentioned services are provided at

http://wiki.wunderground.com/index.php/API_-_XML

http://api.wunderground.com/auto/wui/geo/GeoLookupXML/index.xml?query=94107

http://api.wunderground.com/weatherstation/WXCurrentObXML.asp?ID=KCASANFR70

One major decision regarding the weather stations is that we chose to use only personal weather stations as data sources although Wunderground Data Feed provides data from airports as well. This is because there are many more personal weather stations than airports and these personal stations are scattered around the cities. In addition, there are many places far away from airports

for which weather conditions might not be same as it is at the closest airport. Therefore, reading data from airports might have been misleading.

6.2.3. Integration of Challenge into the Main Game

The integration of the US Challenge into Weatherlings was very quick due to two reasons. First, this challenge is almost completely isolated from the main game. The only shared model is Player, and there are minimal number of operations involving this model. Second and foremost, our experience from integrating the Singapore Challenge into Weatherlings was extremely useful.

6.3. Changes to the Pre-Existing Game

In this section, I will explain the major changes to the pre-existing game. Specifically, I will go over the changes to the database, front-end and back-end. On the other hand, these modifications are not the only ones which took place during the development phase of Weatherlings 2.0. I have maintained the website, by responding to the needs of my team and users and fixing any bugs discovered, which resulted in other changes within the pre-existing game. However, I will not cover every minor detail, and I will only focus on the major ones.

6.3.1. Database

The major change relating to the database is about adding new weather data and weather maps to the game. Initially, there was no method to receive weather data and maps in real-time. Historical weather data was pulled from NOAA National Severe Stroms Laboratory's Historical Weather

Data Archive (http://data.nssl.noaa.gov) and weather maps were scraped by John Pickle and transferred into the game manually. During the development of Weatherlings 2.0, initially this same approach was used, however this is a very inefficient method and we wanted to receive hourly weather data in real-time.

Then, a module, RtWeatherDataRetriever, was written to scrape weather maps in real-time from Weather Underground's website. For instance, http://icons-pe.wunderground.com/data/images/sw_st.gif contains the current temperature map for the South West. For 21 different types of maps and locales, the CRON scheduler calls this module hourly and saves the maps into appropriate folders. On the other hand, our system does not scrape weather data in real-time yet.

6.3.2. Back-End

The most important update with the back-end of the system is the Data Completeness Check which is both integrated into the game and offered as stand alone library tool. The stand alone tool is a module, which goes through all the weather maps in a folder and finds out the missing weather maps for a specified period of time. It can be run manually with the following command:

```
ruby script/runner DataCompletenessChecking.run_data_completeness_check input1 input2 input3
```

The first input is the path of the master folder that contains all the folders which contain the weather images. (there are 21 different folders, such as US_fronts, SW_temperature, ...) The second and third inputs are the start and end dates of the completeness check in YYMMDD format. All the inputs need to be valid for starting the completeness check. For instance, a start date of 091301 will cause an error since the date is invalid due to the month being the 13th

month. For the missing maps, this module reports the following information: locale, type, date and hour (ex: MW, satellite, 091010, 11).

The integrated data completeness check was necessary because the system did not check if relevant maps existed when a battle was started and this resulted in displaying broken links for non-existing maps in the game. The integration of the completeness check makes sure that, while creating a new battle, the battle start time is chosen so that all the weather maps corresponding to all the round times exist. To explain in more detail, a start time is selected at random and all corresponding round times are checked to determine if their weather maps exist. If they do, that is the official start time. Otherwise, another start time is selected at random and the process is repeated until a qualifying start time is found.

CreateMichiganPlayers is another major module which allows mass creation of players. This module is written for the Michigan test study in order to save class time. Instead of signing up in the class, students were handed out their already created account info. This approach proved to be very efficient for the Michigan test study, and in fact, it can easily be used/modified for other studies/games as well. To explain the implementation details of the module, a CSV file, which contains the students' information, is received from the instructor and inputted to the main function of the module. Each student is a new row in the file and the format of a line is:

Firstname,Lastname,Username,Password,Group

Then, student info is parsed, and for each student, the same steps for registering as a new user are taken: a User and a Player objects are created, and an initial deck is created for the Player. The game id of the Michigan game is used in this specific case.

One last modification related to the back-end was allowing users of level 1 to buy cards of rank 1. This makes sense because users of level 1 might want to buy cards of rank 1 instead of rank 0. In addition, this fixes an inconsistency issue since users of levels higher than 1 were already allowed to buy cards of their own level. Other game balancing modifications were applied according to the user comments from user testings.

6.3.3. Front-End

There are only minor changes to the front-end. For example, the homepage icons are updated and re-organized (Figure 26). First concern was that a link to the Singapore/US Challenge had to be added. In addition, headquarters link was very infrequently used and the user name on the bottom of the screen was already a link to headquarters. Therefore, headquarters icon was removed and a link to it was added just below the home page icons (Figure 26b). For the resulting four important links which qualified for having icons, the order of importance was determined as lobby, trunk, challenge and shop; and the order on the screen was updated correspondingly (Figure 26b).

a) Before **b)** After

Figure 26. Change in the Home Page Screen

The columns in the forecast box in a battle are re-ordered: The 'save' button used to be in the beginning of the row (Figure 27a), but it is now at the end of the row (Figure 27b). This is changed to follow the flow of thought: the user first enters information and then clicks on the 'save' button.

a) Before **b)** After

Figure 27. Change in the Forecast Box in Battle Screen

Lastly, the input fields in the forecast box in Figure 27 were modified to show the most recent forecast data if available by default. This can be considered as a reminder to the users that they made a forecast before.

7. SECOND TEST STUDY IN SINGAPORE

This test study was the same as the previous one in many ways. For this reason, I will not repeat the same lines from the previous study and will mostly discuss the new points involved in this study.

Note: US Challenge was not integrated into Weatherlings at the time of this test play.

7.1. Methodology

The methodology was almost exactly the same as the previous one except for the school choice, which turned out to be a major factor in the success of the test study. The test study was conducted in Bedok View Secondary School from April 19th to April 22th 2009. The participation was voluntary and not-graded. Twenty students were selected by the teacher from a group of voluntary students in a classroom of forty. The other details can be found on ‘Section 5.1. Methodology’.

7.2. Findings and Discussion

After the test study was completed, database records corresponding to this study were analyzed through SQL queries. Now, I will go into details of our findings and discuss the results. I will focus on the gameplay patterns observed and peer-to-peer interaction. Then, I will talk about the techniques I have used to extract data from the database, followed by a discussion of the test study.

7.2.1. Gameplay Patterns

The first piece of information extracted from the database was the general numbers related to the gameplay patterns. Table 1 indicates that participants engaged in 45 battles in total, however only 19 of them (42% of all) were completed. 45 created battles corresponded to 224 rounds which meant an average of 5 rounds per battle. 19 completed battles corresponded to 163 rounds which meant an average of 8.6 rounds per battle.

Table 4. Battle and Battle Round Information (all battles include completed and uncompleted battles)

	All battles	Completed battles
Number of battles	45	19
Number of battle rounds	224	163
Average number of rounds per battle	5	8.6

Figures 28 is most related to student interest in the game. Unfortunately, the number of battles initiated decreases every day, 16 to 11 to 10 to 8, which signals declining student interest. Figure 29 shows the predictions made for each day. According to Figure 29, although the first day was a success with 48 predictions, it seems that a lot of students lost interest in forecasting after the first day.

Although the most accurate way to check if play time interfered with the class time would be to analyze the server logs, battle initiation times should give a good enough representation. Figure 30 shows the number of battles initiated in each hourly period over four days. Although students were explicitly instructed to play the game outside the class time, Figure 30 shows that some battles were created during the class time. The battle creation times were not clustered for any

period of time, unlike in the first test study; and the battles were distributed close to evenly from 6am to 23pm.

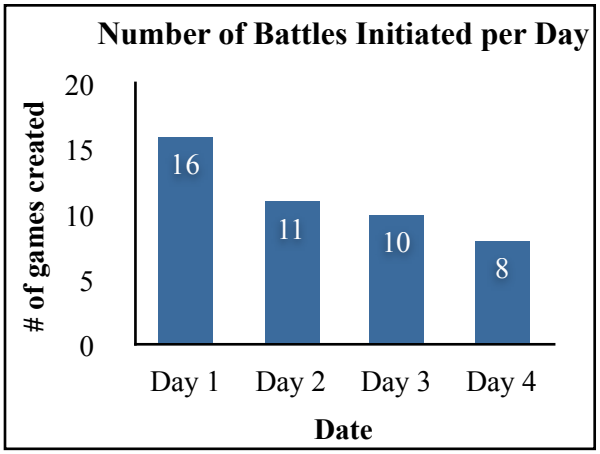


Figure 28

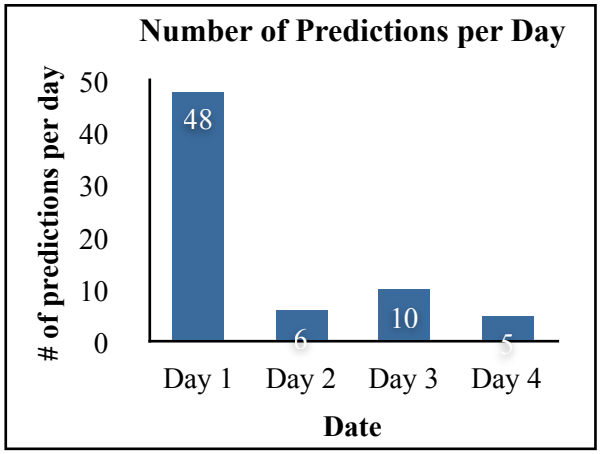


Figure 29

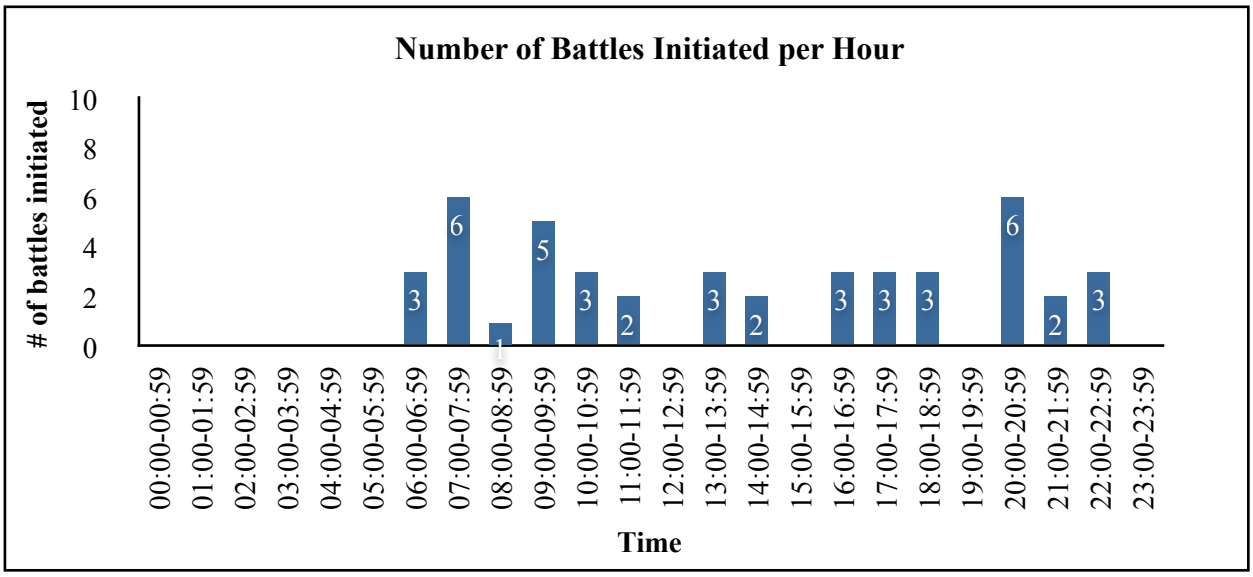


Figure 30

Next, we evaluated the patterns in the Singapore Challenge. Over the four days, there were 35 Singapore Challenges created and 48 forecasts were made, meaning that there were 1.4 forecasts per challenge (Table 5). Figures 31 and 32 show that the interest for the Singapore Challenge

was the highest in the first day, and unfortunately decreased constantly over four days. Figure 33 indicates that the students engaged in the Singapore Challenge mostly after 3pm, and the peak hour was between 5pm and 6pm.

Table 5. Singapore Challenge and Challenge Forecasts Information

Number of Singapore Challenges	35
Number of Singapore Challenge forecasts	49

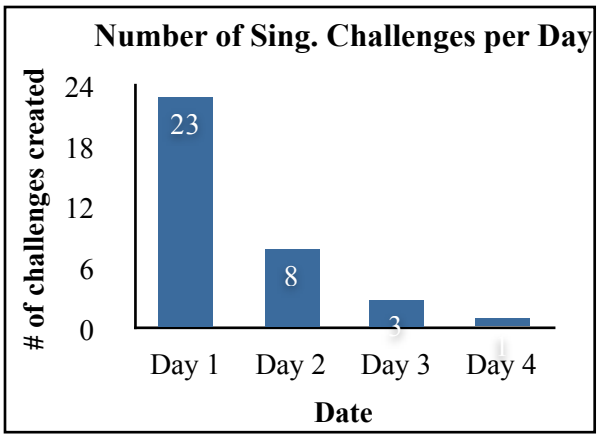


Figure 31

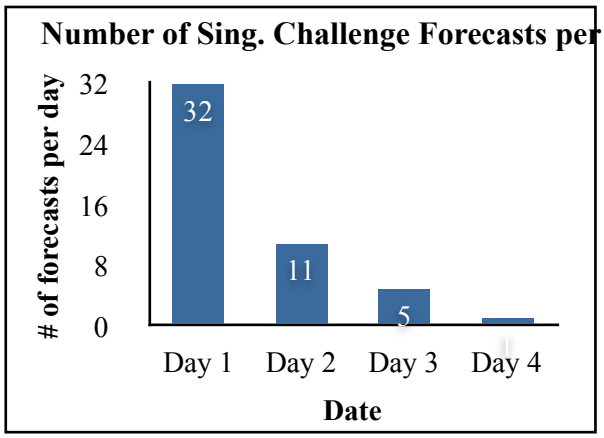


Figure 32

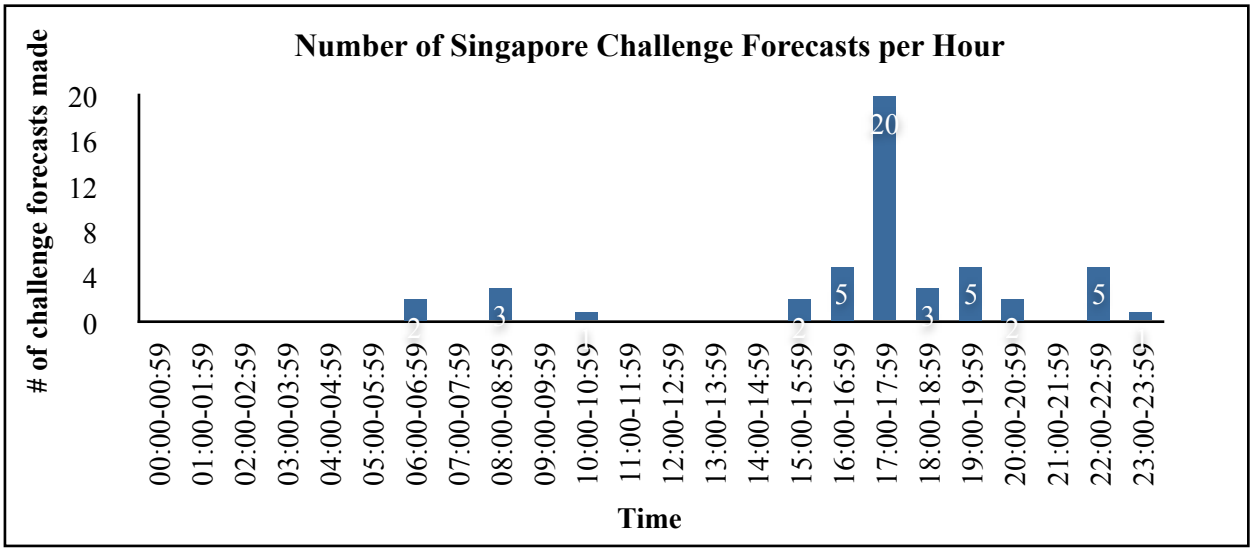


Figure 33

Another very important note is about some students not playing the game at all. Instructors made sure that everyone created an account before the end of the session on Day 1 where the students were given instructions about the game and handed out mobile phones. However, our records indicate that 7 out of 20 students (35%) never logged in after the session. For the 13 students who did actually play the game, Figure 34 shows the number of battles each student engaged in. It seems that 2 students were playing the game intensively (25 and 19 battles), and 5 others were playing somewhat frequently (between 6 and 8). The remaining 6 students were engaged in less than 3 battles.

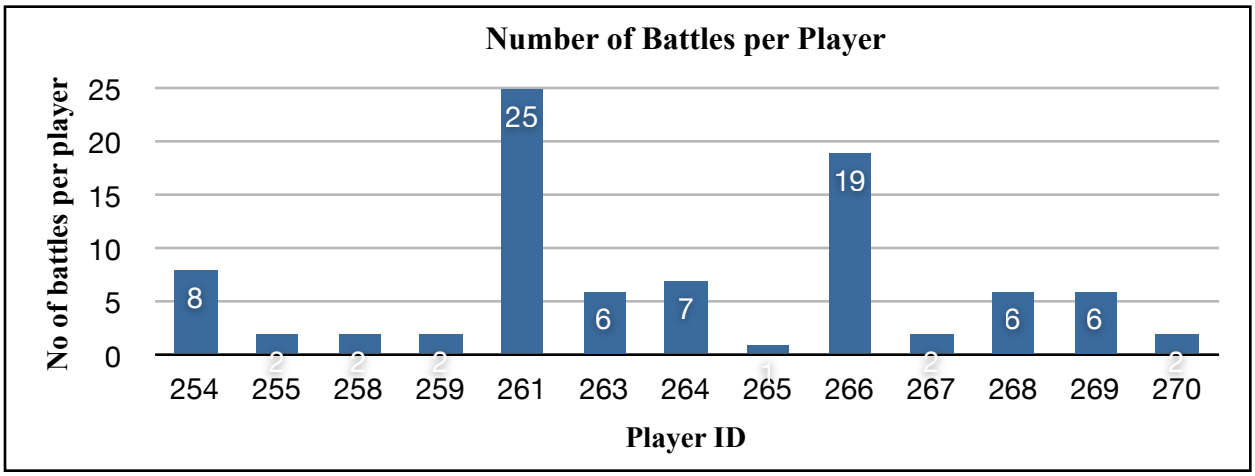


Figure 34

On average, each student played 6.8 battles on average with a standard deviation of 7.26. However, the 2 students with ids 261 and 266 increased the average by a lot just by themselves.

5.2.2. Peer-to-Peer Interactions

Further database analysis for the 13 students who actually played the game indicated that students played against 2.5 unique players on average (SD=1.5). 38% of the 13 students played

against at least 3 (23% of all participants) unique students whereas 31% of them only played against 1 (8% of all participants) unique student (Figure 35). Among all the pair-ups, one of them clearly stands out: students with ids 261 and 266 played 17 battles against each other. These students were the 2 students who played intensively.

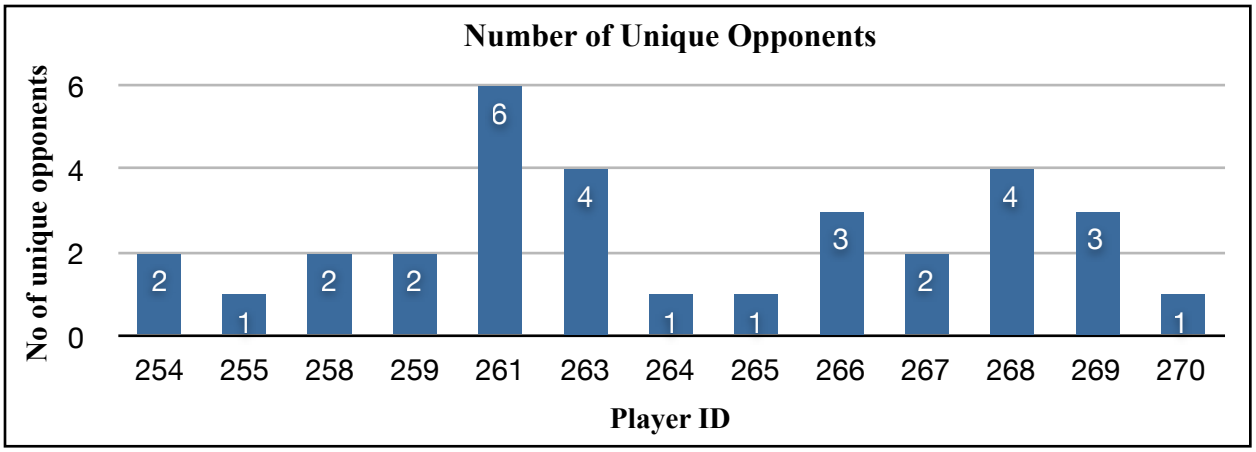


Figure 35

5.2.3. Discussion

As indicated by the results above, unfortunately, the test study was not as successful as we hoped it to be. Relative to the first test study, students were much less interested in the game: 7 of them did not even engage in any battles or challenges at all.

In the discussion with the students at the end of the study, we investigated the possible explanations to why the level of interest was low. It was discovered that some students went directly on Facebook and Youtube as soon as they received the phones and did not want to play the game. Some others said that when they went on to the game lobby, they could not find other players so they decided not to play. One student said that she forgot her password and she did not

know that she could create a new account. We suspect that there might be more students who had technical issues; however, we did not have a chance to talk to each student individually, and most students did not seem to want to share their experiences with the game in front of the entire class.

The teacher told us another possible theory. She said that it was very close to their exam periods, and the parents might have limited the use of the phones at home. And, when the students found a chance to use the phones, they chose to visit other sites such as Facebook which are more familiar to them than our game is.

The instructors of the test study who were present at both test studies and who went to secondary school in Singapore also brought up a possible theory. The previous school, Pei Hwa Presbyterian Primary School, where the study was conducted was one of the top schools in the country, and the students were eager to follow instructions given to them. However, Bedok View Secondary School is not one of the best schools and Bedok's students' ability to follow instructions is not as good as the students from Pei Hwa Presbyterian Primary School is. This theory is also supported by the fact that students from Bedok View Secondary School played some battles in the class time although otherwise instructed, whereas students from Pei Hwa Presbyterian Primary School followed the instructions and did not play in class.

On the other hand, most students indicated that they liked playing the game. When asked what they liked about the game, the most common response was 'the idea of fighting against others'. In addition, some students also thought that they learned about the weather by playing the game.

5.2.4. Information Extraction from Database

The database structure behind Weatherlings is shown in Figures 10, 18 and 24. The information extraction method was the same as the method from the previous study. The SQL queries were very similar to the ones from before such as the following ones.

```
SELECT A1. *
FROM battles A1
WHERE A1.game_id =12
```

```
SELECT A3.id, A2.created_at
FROM battle_rounds A1, battles A2, forecasts A3
WHERE A2.game_id = 12
AND A1.battle_id = A2.id
AND (A1.forecast1_id = A3.id
OR A1.forecast2_id = A3.id)
AND A3.temp IS NOT NULL
```

8. POSSIBLE FUTURE IMPROVEMENTS

8.1. Singapore Challenge

In Section *6.1.2. Implementation*, I have discussed the reliability issue with the computers that the weather stations are connected to in the schools in Singapore. I also mentioned that once Intellisys Lab achieves a stable project state where many schools are running these small computers, our database will have many more stations to receive data from. However, this could be just the beginning of a larger improvement. Currently, the station selection for a challenge is random and there are only three stations. But, once every school has their own weather stations set up and running reliably, a student from one school would only receive weather data from their own school's station which would result in more reliable data for the game play. Ideally, location APIs could be used to determine where the student is. However, not all platforms will have GPS integration (ex: desktops) so there could also be an option, such as a drop down menu, to choose a school manually.

8.2. US Challenge

There are many areas which promise further enhancement in the US Challenge, mostly regarding the data transfer with Wunderground. First one is about using weather data from airport stations. Earlier, it was mentioned that our system only receives data from personal weather stations instead of airports. This was a reasonable decision; however, the ideal behavior of the system would be to check all the personal weather stations and airport stations, and then pick the station which is closest to the given zip code. Implementation of the ideal behavior is possible by

- modifying the UsArena model to mark a station as airport or pws,

- writing an intermediate function which decides which station is closest to the entered zip code (could be a pws or an airport),
- modifying the XML parser which parses the Wunderground Data Feed service output so that it can also handle outputs for airports (although APIs for airport stations and personal stations are similar, they are not exactly same).

In addition, Wunderground Data Feed offers services for weather forecasts and alerts. These information can be integrated into the US Challenge to make the game more interesting. Sample URLs for forecast and alert services are http://api.wunderground.com/auto/wui/geo/AlertsXML/index.xml?query=86445 and http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=Chicago,IL, respectively.

Improvements, which are directly related to the users, are also possible for the US Challenge. Specifically, the step of creating a challenge by entering a zip code can be improved to be more efficient. The first idea is to save the zip code as an attribute of the players and create challenges for that zip code by default. Of course, the users would also be given an option to change the stored zip code whenever they want. Second thought is to show an history of previously entered zip codes with the most recent one as default. The third possibility is to automatically detect zip code through the IP address and use this zip code to create a challenge. However, this is relatively harder to implement than other ideas.

As a final note, it is worth mentioning that the US Challenge is highly dependent on Wunderground Data Feed. If their service goes down, our game cannot be played. For this reason, it might be worth to think about a plan B such as receiving weather data from a second data feed. The chances of two data feeds failing at the same time is much lower than only one of them failing.

8.3. General

There is a very important task which could be very helpful for testing how effective Weatherlings is in teaching about the weather, if implemented. This task is finding out the best battle strategies at given times. To elaborate, students make choices of choosing decks, cards and/or attacks during the game. However, we do not know if their choices are the best ones possible. In fact, there is no quantitative scale which would tell us how good their decisions are. If we can evaluate how good students' decisions are, then we could detect if there are considerable improvements in students' decisions as they play the game, which would be a great source for evaluating our game's educational aspect.

9. FRAMEWORK

Weatherlings is developed as a part of a much larger project, UbiqGames, whose mission is to create a framework for educational game development. The idea behind this framework is to collect all the information possible, which could potentially ease the future development of ubiquitous, educational, multiplayer, mobile, web-based, and platform-independent games. Matt Ng created the first version of this framework from his experiences with developing Virus and Weatherlings 1.0. He mentioned in his thesis that the framework included the design decisions, guidelines and features that can be effective in similar future projects (Ng, 2009). In developing Weatherlings 2.0, I have followed the points from the framework he created, and now I will enrich this framework by adding the most important design decisions, guidelines, and features from my experience with developing Weatherlings.

First, I will divide these into the same categories Ng defined in his thesis: System Architecture, Reusable Modules, Front-end Design, and Third-Party Utilities (Ng, 2009). After that, I will explain the details of each category by first mentioning which of the pre-existing elements I have used, and then discussing the new elements from my experience.

System Architecture

Almost all of the pre-existing elements in this category were helpful to me during the development period. Specifically, I have benefited from the following elements: *stateless client, timestamp data, database model as state machine,* and *separate game and control logic*.

One important addition to system architecture is about separating the Weatherlings code. Initially, there were only two components of the code: game and control logic. However, with the addition of real-time communications to online services, a third component, communication logic, appeared. In both Singapore and US Challenges, a considerable amount of effort was saved by separating the communication logic into separate files since it greatly improved readability and modularity.

Another effort-saving element was the utilization of RoR filters in the controller files. The filters, especially the before_filter helped reduce the number of lines in the controllers and improved readability.

Reusable Modules

These modules were specifically created for the purposes of Weatherlings so they cannot be re-used in their current state. However, they can easily be modified to fit other projects' purposes.

The pre-existing framework included *XML Parsers* for different purposes. Similar XML parsers were written for creating MiniArenas from a file and for creating different kinds of weather datas from service outputs. Two different XML processing libraries were useful: REXML and Hpricot. In addition, one CSV parser was written in a similar fashion to the XML parsers. The CSV parser was necessary for mass-creation of users to Weatherlings. We received the file which contains the users' information in excel format, and the most efficient way to process it was to convert it

to CSV. This makes sense because excel is more commonly used than XML in the non-engineering world.

Apart from parsing, three modules were written to retrieve data from web. For the Singapore Challenge, this communicator file is a web client which uses Simple Object Access Protocol. For the US Challenge, it is a data feed retriever, and for the main game, it is an image grabber. If there is any need to access data from web, these modules can be very useful.

Front-end Design

Although my experience did not result in any new elements for front-end design, many pre-existing elements in this category helped me in the development process. For instance, using modal overlays, link areas, grouping together refreshable sections and screen width were some of the crucial pieces of the UI design.

Third-Party Utilities

Many third-party libraries are used in Weatherlings. I have made use of libraries such as Hpricot and Will_paginate which already existed in the framework. I have also made use of some new libraries, which are explained below.

- Datanoise-actionwebservice: provides SOAP services support for Ruby on Rails.
- Rufus-scheduler: schedules jobs
- REXML: parses XML

In addition to third-party libraries, I have used some applications such as Jeos VM or Jellyfish for development purposes. Since I did development only on Mac OS X, I will suggest these applications for Mac users. However, they should be available for other platforms as well. The full list of these applications is as follows:

- VMware Fusion: We had issues with duplicating the production environment on development machines. Therefore, we took a snapshot of the production machine and set up a virtual machine running it.
- Eclipse with RadRails plug-in: Eclipse is a commonly used IDE. RadRails is a Ruby on Rails development environment.
- Quantum DB: This is an Eclipse plug-in to access databases.
- JellyfiSSH: This provides an efficient way to ssh to remote computers. In my case, I have used it to connect to the production machine and the virtual machine.
- Cyberduck: This is a GUI for FTP/SFTP connections to other computers. I have used it to connect to the production machine and the virtual machine just like JellyfiSSH.

Lastly, CRON scheduler is used to schedule jobs on the production machine, which runs Ubuntu JeOS.

APPENDIX A: Singapore Weather Data Web Service

End Point: *http://nwsp.ntu.edu.sg:8080/axis/EScienceRTDataService.jws*

Function Details

getLatestData – return last data retrieved from a particular station.

Input : <stationid>

Output: <XML: latest weather data collected from the station>

Output Sample:

```
<Result>
        <Station id="RI">
                <Status>Online</Status>
                <Sensor id="Temperature">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>26.9</SensorValue>
                        </Value>
                </Sensor>
                <Sensor id="Humidity">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>90.0</SensorValue>
                        </Value>
                </Sensor>
                <Sensor id="WindSpeed">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>0.0</SensorValue>
                        </Value>
                </Sensor>
                <Sensor id="Rain">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>0.000</SensorValue>
                        </Value>
                </Sensor>
                <Sensor id="SolarRadiation">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>0</SensorValue>
                        </Value>
                </Sensor>
                <Sensor id="Barometer">
                        <Value>
                                <Time>2010-05-12T04:45:00.0</Time>
                                <SensorValue>1012.200</SensorValue>
                        </Value>
                </Sensor>
        </Station>
</Result>
```

APPENDIX B: Full List of Libraries Used in Weatherlings 2.0

actionmailer (2.3.5)		hpricot (0.8.2, 0.8.1)
actionpack (2.3.5)		json_pure (1.2.0)
activerecord (2.3.5)		linecache (0.43)
activeresource (2.3.5)		lockdown (1.6.3)
activesupport (2.3.5)		mislav-will_paginate (2.3.11)
authlogic (2.1.3)		packet (0.1.15)
builder (2.1.2)		polyglot (0.2.9)
calendar_date_select (1.15)		rack (1.0.1)
cheat (1.2.1)		rails (2.3.5)
chronic (0.2.3)		rake (0.8.7)
columnize (0.3.1)		rmagick (2.12.2)
cucumber (0.6.2)		ruby-debug (0.10.3)
datanoise-actionwebservice (2.3.2)		ruby-debug-base (0.10.3)
diff-lcs (1.1.2)		rubyforge (2.0.3)
fastercsv (1.5.3)		rubygems-update (1.3.5)
fastthread (1.0.7)		rufus-scheduler (2.0.4)
gemcutter (0.3.0)		term-ansicolor (1.0.4)
gruff (0.3.6)		treetop (1.4.3)
haml (2.2.19)		validates_timeliness (2.3.0)
hoe (2.5.0)		will_paginate (2.3.12)

APPENDIX C: MiniArena data XML snippet

```
<mini_arena>
        <name>NTU Intelligent Systems Centre (V)</name>
        <city></city>
        <state></state>
        <station_code>intellisys(v)</station_code>
        <climate></climate>
        <elevation></elevation>
        <latitude>1</latitude>
        <longitude>104</longitude>
        <description></description>
</mini_arena>
<mini_arena>
        <name>NTU Intelligent Systems Centre (W)</name>
        <city></city>
        <state></state>
        <station_code>intellisys(w)</station_code>
        <climate></climate>
        <elevation></elevation>
        <latitude>1</latitude>
        <longitude>104</longitude>
        <description></description>
</mini_arena>
```

APPENDIX D: GeoLookupXML API Snippet

URL: http://api.wunderground.com/auto/wui/geo/GeoLookupXML/index.xml?query=94107

```
<location type="CITY">
        <country>US</country>
        <state>CA</state>
        <city>San Francisco</city>
        <tz_short>PST</tz_short>
        <tz_unix>America/Los_Angeles</tz_unix>
        <lat>37.76834106</lat>
        <lon>-122.39418793</lon>
        <zip>94107</zip>
        ...
        <nearby_weather_stations>
                <airport>
                        <station>
                                <city>Oakland</city>
                                <state>CA</state>
                                <country>US</country>
                                <icao>KOAK</icao>
                                <lat>37.72</lat>
                                <lon>-122.22</lon>
                        </station>
                        ...
                </airport>
                <pws>
                        <station>
                                <neighborhood><![CDATA[Mission Bay]]></neighborhood>
                                <city><![CDATA[San Francisco]]></city>
                                <state><![CDATA[CA]]></state>
                                <country><![CDATA[US]]></country>
                                <id><![CDATA[KCASANFR53]]></id>
                                <distance_km><![CDATA[0]]></distance_km>
                                <distance_mi><![CDATA[0]]></distance_mi>
                        </station>
                        <station>
                                <neighborhood><![CDATA[China Basin]]></neighborhood>
                                <city><![CDATA[San Francisco]]></city>
                                <state><![CDATA[CA]]></state>
                                <country><![CDATA[US]]></country>
                                <id><![CDATA[KCASANFR14]]></id>
                                <distance_km><![CDATA[1]]></distance_km>
                                <distance_mi><![CDATA[0]]></distance_mi>
                        </station>
                        ...
                </pws>
        </nearby_weather_stations>
</location>
```

APPENDIX E: WXCurrentObXML API Snippet

URL: http://api.wunderground.com/weatherstation/WXCurrentObXML.asp?ID=KCASANFR70


```
<current_observation>
	<location>
		<full>Sunnyside, San Francisco, CA</full>
		<neighborhood>Sunnyside</neighborhood>
		<city>San Francisco</city>
		<state>CA</state>
		<zip/>
		<latitude>37.732376</latitude>
		<longitude>-122.442902</longitude>
		<elevation>100 ft</elevation>
	</location>
	<station_id>KCASANFR70</station_id>
	<observation_time>Last Updated on January 15, 9:22 AM PST</observation_time>
	<observation_time_rfc822>Tue, 15 January 2008 17:22:35 GMT</observation_time_rfc822>
	<temperature_string>44.8 F (7.1 C)</temperature_string>
	<temp_f>44.8</temp_f>
	<temp_c>7.1</temp_c>
	<relative_humidity>97</relative_humidity>
	<wind_string>Calm</wind_string>
	<wind_dir>NNW</wind_dir>
	<wind_degrees>338</wind_degrees>
	<wind_mph>0.0</wind_mph>
	<wind_gust_mph>1.0</wind_gust_mph>
	<pressure_string>30.30" (1026.0 mb)</pressure_string>
	<pressure_mb>1026.0</pressure_mb>
	<pressure_in>30.30</pressure_in>
	<dewpoint_string>44.0 F (6.7 C)</dewpoint_string>
	<dewpoint_f>44.0</dewpoint_f>
	<dewpoint_c>6.7</dewpoint_c>
	<heat_index_string/>
	<heat_index_f/>
	<heat_index_c/>
	<windchill_string/>
	<windchill_f/>
	<windchill_c/>
	<precip_1hr_string>0.00 in (0.0 mm)</precip_1hr_string>
	<precip_1hr_in>0.00</precip_1hr_in>
	<precip_1hr_metric>0.0</precip_1hr_metric>
	<precip_today_string>0.00 in (0.0 mm)</precip_today_string>
	<precip_today_in>0.00</precip_today_in>
	<precip_today_metric>0.0</precip_today_metric>
	<history_url>http://www.wunderground.com/weatherstation/WXDailyHistory.asp?
ID=KCASANFR70</history_url>
	<ob_url>http://www.wunderground.com/cgi-bin/findweather/getForecast?
query=37.732376,-122.442902</ob_url>
</current_observation>
```

REFERENCES

Colella, V. (2000). *Participatory simulations: Building collaborative understanding through immersive dynamic modeling.* The Journal of the Learning Sciences 9:471-500.

Gee, J. (2003). *What video games have to teach us about learning and literacy.* New York:Palgrave.

Klopfer, E. (2008). *Augmented Learning: Research and design of mobile educational games.* Cambridge, MA: The MIT Press.

Klopfer, E., Sheldon J., Perry J., Chen, Dr. Vivian Hsueh-Hua (2010). *Ubiquitous Games for Learning (UbiqGames): Weatherlings, a Worked Example.* Not published yet, in review.

Martinez, S. (2006). *Educational Games: How Attitudes and Markets Influence Design.* Australian Computers in Education Conference

Mitchell A. and Savill-Smith, C. (2004). *The use of computer and video games for learning. A review of the literature.* Learning Skills and Development Agency, Ultralab, London, UK

Negroponte, Nicholas. (2006). *The vision of One Laptop Per Child.* TEDTalks, Monterey, CA, February 2006. Available online at http://video.google.com/videoplay?docid=-2454177915360000762&ei=K2ogS-bvI4KurQLXq4wU&hl=en#

Ng, Matthew. (2009). *UbiqGames: Casual, Educational, Multiplayer Games for Mobile and Desktop Platforms.* Thesis of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology, Cambridge MA

NOAA National Severe Stroms Laboratory. Historical Weather Data Archive. http://data.nssl.noaa.gov

Prensky, M. (2001). *Digital game-based learning.* New York: McGraw Hill.

The National Weather Study Project (NWSP). http://nwsp.ntu.edu.sg/weather/main_services.htm

Weather Underground, Inc. (2010). WUNDERGROUND DATA FEED. http://wiki.wunderground.com/index.php/API_-_XML http://icons-pe.wunderground.com/data/images/sw_st.gif