threephase — A Browser-based Power System Simulation Game

Christopher Peplin (peplin@cmu.edu)
Advisor: Dr. Jay Apt, Advisor (apt@cmu.edu)
Advisor: Dr. Gabriela Hug, Advisor (ghug@ece.cmu.edu)

May 24, 2011

Abstract

threephase is a web browser-based serious game, or simulation, of the electric power generation and transmission system. The power system is a growing, popular concern of which the complexity is not well understood by non-experts. The simulations and teaching tools currently available aren't sufficiently accessible and modern to attract people from outside the industry. **threephase** balances between the artistic, the playful and the technical to create an immersive virtual world for experimentation and learning.

From conception to implementation, the design shifted in a few ways in respose to the demands of the web-based user interface. The nature of the web protocol HTTP also presented unique challenges to a real-time game, and **threephase** applies some novel techniques to find scalable solutions.

The code powering the game is provided under the MIT open source license at `https://github.com/peplin/threephase` and continues as a community driven project.

This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.

Contents

0.1 Preface . . . IV
0.2 Acknowldgements . . . IV
0.3 Source Code . . . IV

1 Introduction **1**
1.1 Computer & Video Games . . . 1
1.2 Concept . . . 3

2 Game Objects **5**
2.1 Nomenclature . . . 5
2.2 Game/Country . . . 5
2.2.1 Attributes . . . 5
2.2.2 Map . . . 6
2.3 Multiplayer Elements . . . 6
2.4 State . . . 7
2.4.1 Average Cost Curve . . . 7
2.4.2 Geographic Visualization . . . 8
2.4.3 Location Based Discounts . . . 8
2.5 Creating Objects . . . 9
2.5.1 Generator Type Attributes . . . 9
2.5.2 Object Type Extensibility . . . 10

3 Updating the Game State **12**
3.1 Update Frequency . . . 12
3.1.1 On-Demand Updates . . . 12
3.1.2 Crossing the Day Boundary . . . 12
3.2 Update Scope . . . 13
3.2.1 Distributed Updates . . . 13
3.2.2 Accelerated Game Speed . . . 14
3.2.3 `GameTime` Helper . . . 14

4 Player Motive **18**
4.1 Demand . . . 18
4.1.1 Changing Demand . . . 18
4.2 Primary Player Goals . . . 19
4.2.1 Profit . . . 19

5 Technical **22**
5.1 Time Distribution . . . 22
5.2 Test-driven Development . . . 23
5.3 Graphics . . . 24
5.4 Database . . . 24
5.5 Asynchronous Tasks . . . 24

6 Evaluation **26**
6.1 Criteria . . . 26
6.2 Design Changes . . . 26

7 Summary & Future Work **27**
7.1 Serious Games . . . 27
7.2 Current Player Actions & Abilities . . . 27
7.3 Improvements . . . 28
7.4 Open Source . . . 29

0.1 Preface

This report presents the implementation strategy and game mechanics of **threephase** , the challenges encountered and the future plan for the system. The project was completed over a two month period beginning in September 2010. It was completed to satisfy the graduate project requirement for Master's in Information Networking (MSIN) candidates at the Information Networking Institute of Carnegie Mellon University (CMU).

The primary inspiration for **threephase** was the class "The Engineering & Economics of Power Systems" offered at CMU in the Spring of 2010 [9]. The class introduced the core concepts of the power system, and discussed many of the issues effecting the utilities today. The available computer simulations for learning these concepts had room for expansion and improvement.

This report also serves as a call for contributions from those interested in serious games, the power system, web application development, user interfaces and design. **threephase** will be gradually expanded to cover other areas of the power system, some of which are mentioned here.

0.2 Acknowldgements

I wish to thank the following faculty from Carnegie Mellon University who taught the aforementioned class in power systems [9] that sparked my curiosity in power systems. They are:

- Dr. Marija Illic
- Dr. Jovan Illic
- Dr. Lester Lave

I also wish to thank Lauren Fleishman, a Ph.D. candidate at Carneige Mellon University, who provided me with educational materials she created to inform the public about different power generator types.

0.3 Source Code

The source code powering the game is provided under the MIT open source license at `https://github.com/peplin/threephase`.

List of Figures

1.1 Screenshot from *SimCity 2000*. 2
1.2 Screenshot from *Darwinia*. 2
1.3 Screenshot from *Love*. 3
1.4 Screenshot from *Gipsys*. 4
1.5 Screenshot of the website for IBM's serious game CityOne. 4

2.1 Class diagram of proximity of a relationship between game objects. 6
2.2 Screenshot of graph of average cost curve for generators. 7
2.3 Screenshot of State "heads-up display". 8
2.4 Screenshot of in-game map. 9
2.5 Class diagram of *TechnicalComponent* hierarchy. 11

3.1 Chart of decreasing rate of updates compared to an even timestep. 13
3.2 Screenshot of marginal price graph. 14
3.3 Representation of naive, update-all approach to game updating. 15
3.4 Representation of improved, list-based update approach game updating. 16
3.5 Representation of lazy approach to game updating. 16
3.6 Representation of crossing day boundaries between game state updates. 17
3.7 Code example of a GameTime scaled time object. 17

4.1 Screenshot of a *City*'s load profile graph. 19
4.2 Map of cities with physical line constraints. 20
4.3 Map of cities with Locational Marginal Prices. 21

5.1 Graph comparing time distribution of work in the project. 22
5.2 Code example of an RSpec test case. 23

Chapter 1

Introduction

Now more than ever, the consumption and generation of electricity are on the minds of policy makers and concerned citizens alike. Green power, smart grids, and the renewed popularity of nuclear energy seem like obvious solutions to increasing efficiencies, so the lack of implementation momentum puzzles many people outside the industry. The most (and nearly only) visible change in the past decade to consumers is the shift from incandescent to CCFL light bulbs — hardly revolutionary.

Since the widespread restructuring of the power system in the early 1970's, the complexity of power economics has surpassed the understanding of most people, including the politicians charged with deciding the future of the system itself. The engineering problems are also non-intuitive to those without an electrical engineering background. For example, despite the hype, wind power alone is not the ultimate solution to the world's energy and environmental issues, but this isn't communicated to or well understood by laypeople. There is an opportunity for educating the public and increasing awareness of the tough realities of the power system.

1.1 Computer & Video Games

The power system is frequently included in computer and video games, dating at least back to Maxis' *SimCity* of 1989 (Figure 1.1). Electricity appears even earlier in board games, where controlling the power & water utilities in *Monopoly* garnered players a key advantage. More recently, the German board game *Power Grid* [13] used the power system as its core game mechanic.

Electricity transmission surfaced as a game mechanic in recent computer games as well, such as *Darwinia* (Figure 1.2) and the new massively multiplayer game *Love* (Figure 1.3). In both games, protecting transmission lines from attack and malfunction is a key objective.

On the other end of the spectrum, the current power system teaching tool used at Carnegie Mellon University, *Gipsys* (Figure 1.4), excels in the technical but isn't approachable enough to engage those with a passing interest. Since this project started, IBM released a web-based city planning serious game, *CityOne* (Figure 1.5), which asks players to make public policy decisions to improve efficiency in their virtual city. IBM's take on serious games is unfortunately less of a challenging, immersive virtual world and more of a marketing tool.

The frequent appearance of the electrical system in games is not a coincidence — the concepts of generation and transmission fit well with strategy gameplay. The games market is ripe for a serious game that combines popular fascination with an idealized power system and the often troublesome state of engineering and economics in the actual industry. This game could be used

Figure 1.1: Screenshot from *SimCity 2000*, the second release of the SimCity franchise. Power system configuration is limited to generator type and the path of transmission lines [17].

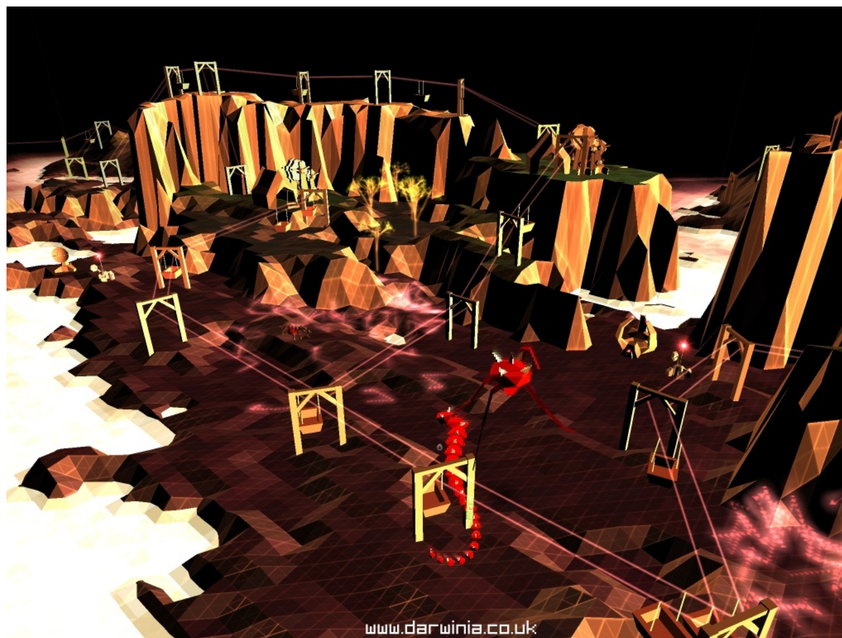

Figure 1.2: Screenshot from *Darwinia*, where game strategies revolve around reconnecting power loads to generators after a pseudo-terrorist attack [4].

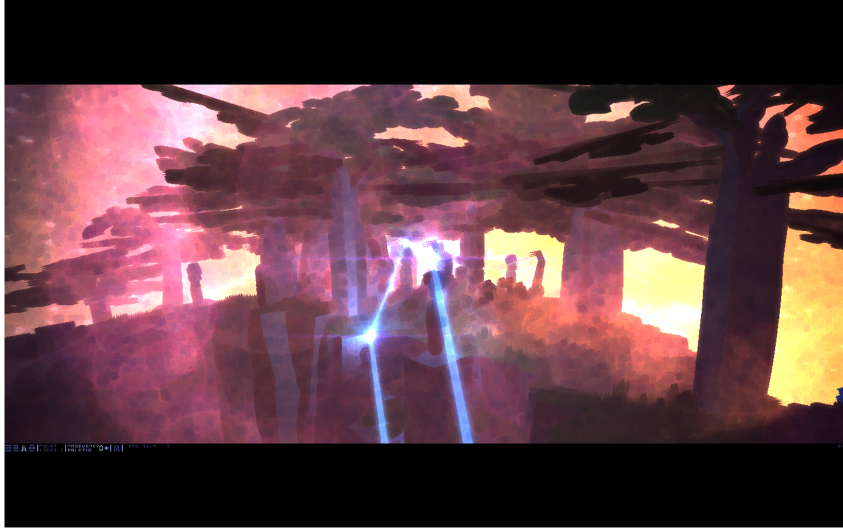

Figure 1.3: Screenshot from *Love*, where every device used to build cities in the game must first be powered by connecting a network of transmission lines from a first-person view [11].

for both education and casual enjoyment.

1.2 Concept

threephase tries fill the remaining gap, and balance between the artistic, the playful and the technical. A new generation of gamers is being formed online, by the likes of Zynga's Farmville, Frontierville and Mafia Wars [22]. These gamers are comfortable with having a persistent, virtual world in the games they play. They are accustomed to games lasting days or months, and even those without a set endpoint. Unfortunately, few of these games challenge players to learn or think creatively. These games are a missed opportunity to show a wide audience the positive effects of gaming firsthand.

The goal of **threephase** is to be approachable by a lowest common denominator of people who understand technology, use the web and are willing to play a game (or already do). Each player is handed control of a state-wide utility company and tasked with generating enough power to meet customer demand. Each player operates in a game world shared with other players, where the repercussions of energy decisions in one state can be felt by many others.

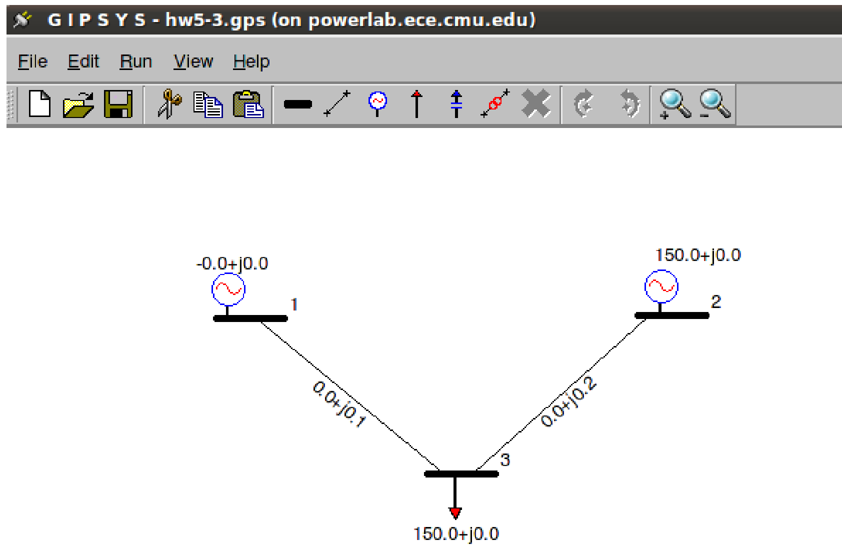

Figure 1.4: Screenshot from *Gipsys*, a simulator focused on technical aspects with minimal presentation [7].

Figure 1.5: Screenshot of the website for IBM's city planning serious game CityOne, released in October 2010 [3].

Chapter 2

Game Objects

2.1 Nomenclature

The primary geographical units of **threephase** are the *Country*, the *State* and the *City* (see figure 2.1). At any time, there are multiple game worlds running simultaneously. Each game (or *Country*) has its own unique attributes and available power technologies. The player can choose to either join an existing *Country* or create a new one (perhaps to experiment with a new regulatory system). A *Country*'s virtual world is ongoing and persistent — players can join and leave at any time.

To join a *Country* (game and *Country* are used interchangeably), the player creates a *State*. The *State* is the player's relationship to a certain *Country* — they can participate in multiple games simultaneously, but with only one *State* in each game. The player assigns a name, research budget (which effects the cost of new technology) and a map to the new *State*.

2.2 Game/Country

In order to lower the barrier to entry for new players, the player may join and leave games at any time. The time spent finding and joining a game should be minimal. The player should be able to start making in-game decisions as soon as possible to grab and keep their attention. The effects of a *State* suddenly dropping out of the *Country* are dampened to not negatively effect the experience for other players when someone leaves the game.

2.2.1 Attributes

A *Country* has many adjustable attributes that are set at creation. These include, but are not limited to:

- Minimum & maximum transmission line capacity
- Relative cost of technology
- Relative wind speed
- Type of economic regulation

The attributes affect the difficulty of the game (e.g. relatively higher capital costs make new construction more difficult) similar to how new laws and public policies can affect power utility

strategy in real life. They can also simulate different physical environments, e.g. the relative scale of the average wind speed can make a *State* more or less inclined to add wind turbines to their generator portfolio.

Economic Regulation Economic regulation is a critical factor in operating a profitable utility. The available economic regulation types in **threephase** are rate of return, marginal cost bidding, and locational marginal pricing (see 4.2.1). Power utility regulation is an unsolved problem, and the ability to switch between types in **threephase** makes comparing scenarios in different regulatory environments a possible use case.

2.2.2 Map

Each *State* has a map which defines the land and natural resources underneath and neighboring each *City*. The proximity of *City* to areas of the map can have a non-trivial effect on the price of certain types of generation in the *City* because of the abundance of natural resources (discussed further in 2.4.3).

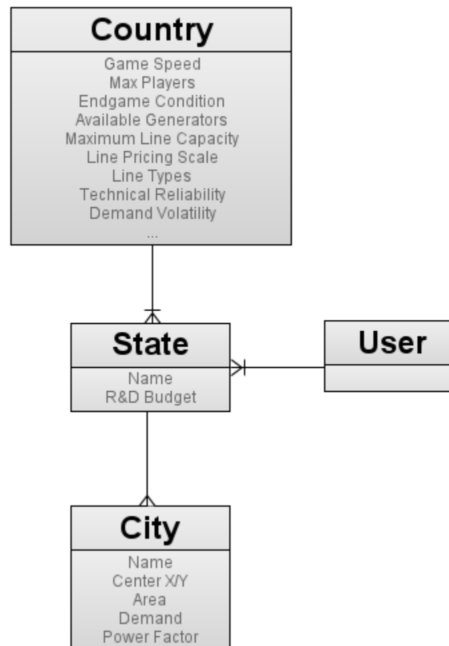

Figure 2.1: A class diagram visualizing the relationship between game instances (a *Country*), States, Cities, and Users. Each user can participate in multiple games at once, but only in one State per game.

2.3 Multiplayer Elements

threephase is a multiplayer game. Each *Country* is shared among the players that control a *State* in that *Country*. These players currently share national fuel prices. Generators that use non-renewable fuel purchase their operating fuel at prices determined by a national fuel market. Every day, the market price for each fuel type (e.g. coal, oil, natural gas, etc.) is cleared based

on the total demand (the sum of the demand of all generators using that fuel, with respect to their projected operating level) and the total supply (the sum of the availability of the fuel's raw natural resource in the *State* maps). To bootstrap the economy, each fuel market is initialized with a starting price (randomized within a pre-specified standard deviation, for variety's sake) and a price elasticity of supply. The supply of fuel does not change since *State* maps are static in the current implementation, so the price elasticity is used to calculate how the fuel price reacts to changes in demand.

The shared fuel prices mean that a player's decision to build many large coal generators can have implications on the generator portfolios of another player (who may be less inclined to build coal plants due to the increasing cost of fuel).

2.4 State

Within a *State*, the player has complete control over all generators and transmission lines. The player can view the average cost curve of their generators (see figure 2.2), graphed in order of their average cost. The ideal, cost-minimizing strategy would run the generators in this order — cheapest first.

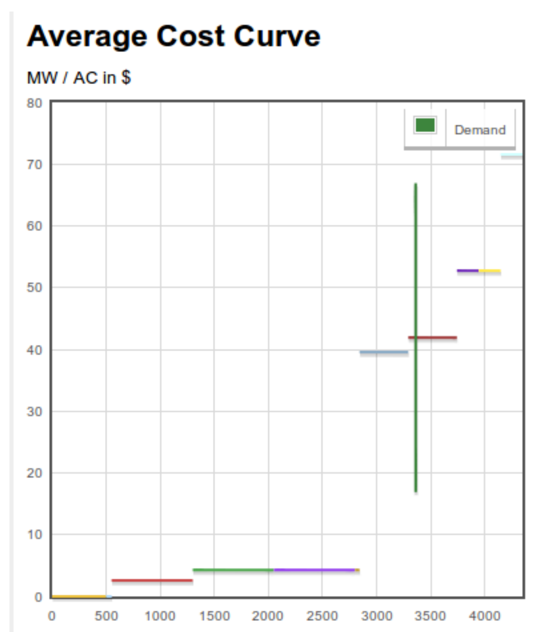

Figure 2.2: Screenshot of graph of average cost curve for generators. Each State has a graph of its generators, ordered by their average cost to produce a MWh of electricity. In the ideal case, generators are run in the order they appear in this curve.

2.4.1 Average Cost Curve

threephase makes a conscious decision to use the average cost curve, as opposed to the marginal cost curve, to attempt to take into the account the capital investment of each generator. This is a deviation from the industry norm, done in part to assist the author's own understanding. A common point of confusion for non-experts is how utilities could ever expect to do more than depreciate their equipment operating at the marginal cost. Different types of regulation

attempt to compensate in various ways, most commonly with what are known as capacity payments. These are side payments made by the regulator to encourage re-investment and continued expansion of operations.

2.4.2 Geographic Visualization

The *dashboard* of each state displays a basic visualization of the *State* and its *Citys* (see figure 2.3). The map also visualizes the composition of the land underneath the *State*. Each small dot is a *Block*, and each *Block* has one of a few different types — mountains, plains or water. Each *Block* also has an index per natural resource, describing its relative abundance in that area. There are currently indices for the non-renewable resources natural gas, coal and oil and for the renewable resources sun, water and wind. In the current implementation, the block types and indices are chosen randomly. This can be improved using map generation algorithms to create more natural and useful land organization — mountain ranges, rivers, etc.

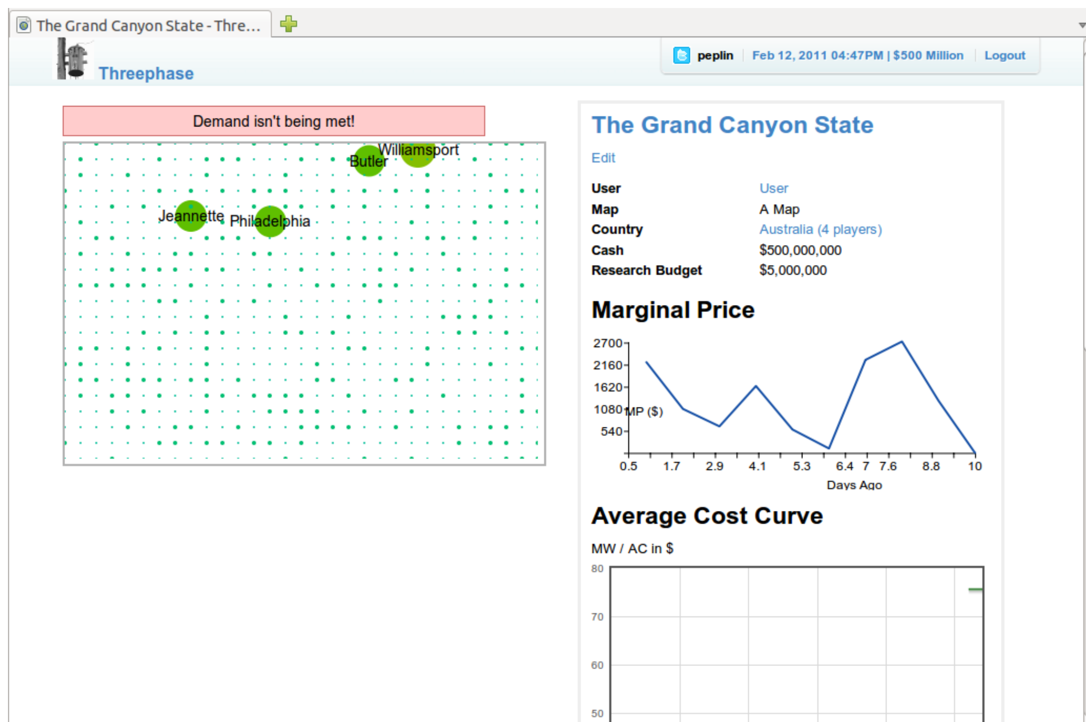

Figure 2.3: A screenshot of the State "heads-up display", which shows a map of the natural landscape, where the cities fall in relationship to natural resources, and historical statistics on the price of fuel and electricity.

2.4.3 Location Based Discounts

The location of a *City* on the map has important implications. Based on the availability of coal within the region, for example, a certain percentage discount is given to coal generators operating in that *City* (see figure 2.4). Wind turbines in an area of *Blocks* with high wind indices will be more effective than in other places.

The discounts are calculated by finding all blocks within a certain radius of the *City*, scaled based on the population. Larger cities will extend further out from their center point, so they

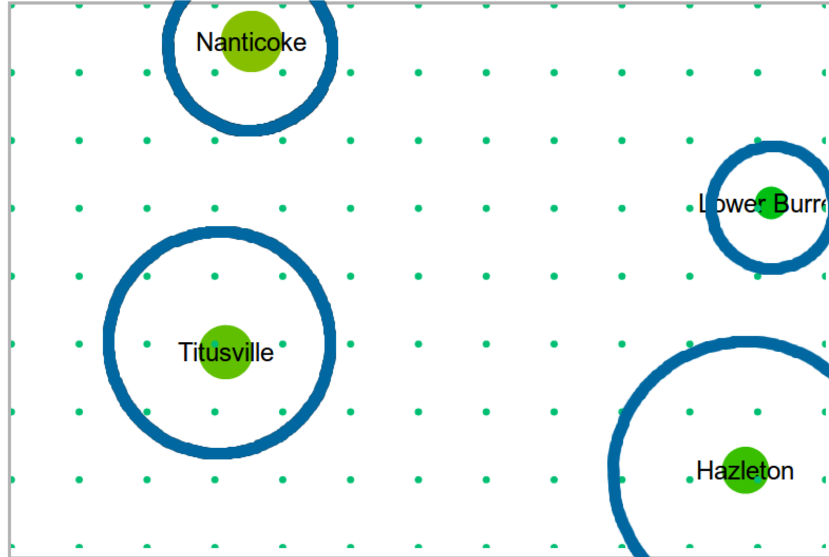

Figure 2.4: A screenshot of the in-game map. The natural resource indices of the land beneath each *City* can have an effect on the price of local generation. Blocks within a certain radius of a *City* (green dots within the blue circle, scaled based on population) that contain large amounts of coal or natural gas can make a *City* a good choice for matching generator types.

can be expected to utilize a wider area of natural resources.

2.5 Creating Objects

To create a generator, the player selects a *GeneratorType* from a list of available technologies. Each *Country* can customize the list of available *GeneratorTypes* to simulate different time periods and regulatory environments (e.g. before advanced nuclear or coal with carbon capture and sequestration). The player can compare the available *GeneratorTypes* based on their attributes to decide which would be the best choice for their portfolio. The current interface is a simple table, sortable by column. Additional comparison visualizations can make the choices easier to understand, and convey some of the issues with new technologies. Lauren Fleishman's work in summarizing and comparing generators is a good inspiration for the user interface [5].

2.5.1 Generator Type Attributes

The attributes for a *GeneratorType* loosely belong to three different groups — those that:

- Scale the marginal cost
- Scale the capital cost
- Scale the rate of occasionally positive & negative one-time events

For example, the *waste disposal cost* effects the marginal cost of power for the *GeneratorType*. The *tax credit* lowers the initial capital investment, relative to the size of the generator. The *technical reliability* effects the frequency of equipment failure, and the *technical*

complexity effects the time it takes to repair a generator once a failure occurs. A complete list is available in the **threephase** wiki [12]. Some of the values for attributes were inferred from the U.S. Energy Information Administraton's website [20], but others still need to be researched.

Capacity Range Each *GeneratorType* also has a range of valid capacities - the number of MWh the *GeneratorType* can produce at its peak. This range reflects the general capabilities of the *GeneratorType*, and also its typical applications in real world power grids. For example, gas turbines have relatively lower capacity limits than nuclear generators, and they are typically used as peaking plants (to cover spikes in demand) as opposed to baseload plants which are more efficient in large capacities.

2.5.2 Object Type Extensibility

These attributes and effects are not specific to generators. The objects in **threephase** are members of a *TechnicalComponent* hierarchy, allowing them to share the flexibility enjoyed by generators. Transmission lines, power storage devices, and other component classes (see figure 2.5) all have these attributes. In the case of transmission lines, the player can choose from *LineType*s such as high-voltage DC and high-voltage AC of varying capacities and resistances. Most of the attributes are shared with generators through the *TechnicalComponent* model, but class-specific attributes (e.g. underground v.s. above ground lines) are also supported.

Implementation Note The relationships are maintained using single-table inheritance (so *GeneratorType*s, *LineType*s and *StorageDeviceType*s share a database table) and polymorphic associations (so a *State* can reference instances of the three classes in a generic fashion).

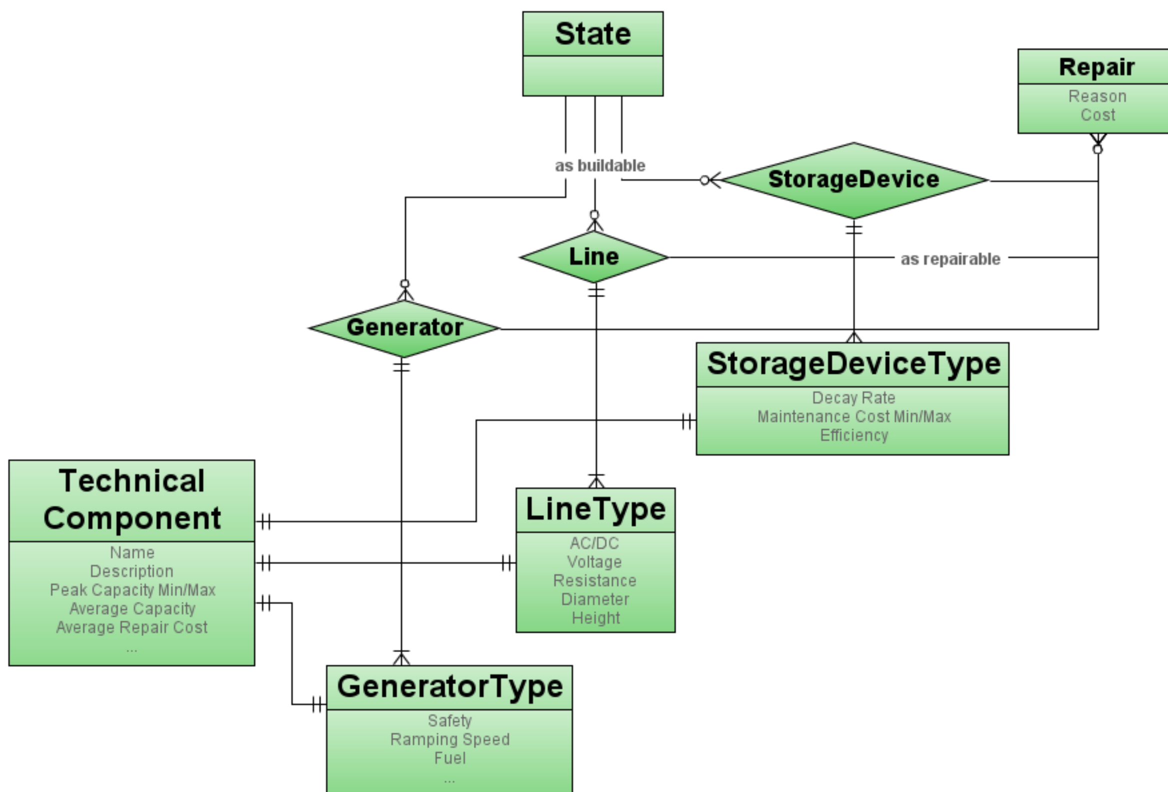

Figure 2.5: A class diagram of *TechnicalComponent* hierarchy. Generators, transmission lines and power storage devices all inherit from a common parent type — the Technical Component. This allows the game to share code when interacting with each type of object, and still allow for some customization.

Chapter 3

Updating the Game State

A primary task for any game is making things move. The default approach in a single-player, desktop computer game is somewhat clear. The software has total control over the CPU, and there is only a single player waiting for game world updates. A web-based game needs to optimize for many simultaneous games, while minimizing server costs.

3.1 Update Frequency

In a single-player, desktop computer game, it is reasonable to set a small timestep (e.g. 1 second) at each of which the game will recalculate and update the entire world. Powerful processors permit this method to scale up to very large games locally, providing the player with a consistent, up-to-date world.

Unfortunately, this strategy doesn't scale to the web, where a server will have more individual game instances to process and where it is infeasible to dedicate an entire processor to a single game instance at all hours of the day. Thanks to the patterns of web users, there are some relatively easy ways to decrease the amount of work that needs to be done. Updating at a regular timestep would be overeager, when a player typically isn't visiting the web application every second or demanding real-time updates. Users are much more tolerant of short delays (i.e 1-5 seconds) on the web than in desktop applications.

3.1.1 On-Demand Updates

At the minimum, **threephase** updates on demand, only when the player visits their game. To provide background, out-of-band notifications (e.g. e-mails with in-game alerts), however, the game does occasionally need to be updated to check the conditions. Instead of maintaining a steady, short timestep, **threephase** gradually decreases the interval between updates as a player stops visiting the game (see figure 3.1). 12 hours after a visit, the game updates only once an hour. Three days after the last player visit, game updates may be as infrequent as once a day. Players visiting every day (and by implication those more invested in their virtual world) will receive an appropriately higher update rate, thanks to the processing power left free by those visiting **threephase** less often.

3.1.2 Crossing the Day Boundary

The technical impact of this design is that every element of the game needs to be able to update itself over any time interval — 10 seconds, 10 minutes, 10 days. Instead of simply calculating the

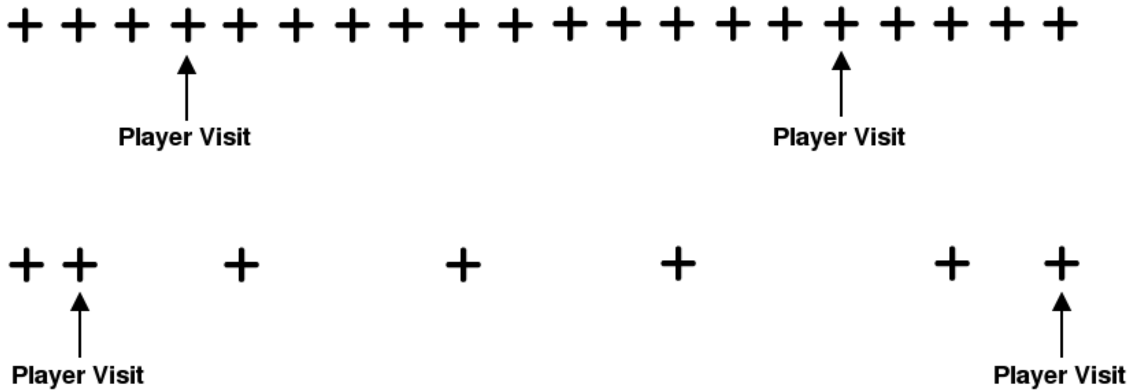

Figure 3.1: A chart of decreasing rate of updates compared to an even timestep. Constant timestep updates are inefficient in an online game, as players will not need updates most of the time. If a player is not requesting information, the data doesn't not need to remain up-to-date. Between player visits, the update timesteps (i.e. the crosses in the figure) can be slowed and even stopped.

difference between now and 1 second ago, the objects need to handle updates spanning multiple days. This is important in **threephase** because there are certain statistics and calculations that need to occur exactly once per day. That includes:

- Calculating the marginal cost curve
- Clearing fuel market prices
- Storing average marginal price and operating level statistics

The algorithms to calculate these values need to be able to step through the calculation for multiple days, and not condense the change into a single value. For example, if three days have passed since an update, the server must to calculate the marginal price of power on each of the three days, not on average (see figure 3.2). It is possible to use an integral in some situations where the calculations per day are unimportant, and this method is preferred if possible.

3.2 Update Scope

In addition to the question of when to update, the game must decide how much to update. A naive approach is to update every game element, every time, starting from the top of the object hierarchy (see figure 3.3). This method grows in complexity exponentially as games are created, and does a lot of unnecessary work. It also runs into problems with a fixed timestep — if the work cannot be finished before the next interval (e.g. 1 second), the updates will fall behind and never catch up. Another method is to maintain a list of only the objects that explicitly need to be updated (see 3.4). This avoids duplicated work, but is difficult to maintain.

3.2.1 Distributed Updates

threephase uses a distributed update strategy, where updates start at the lowest levels of the hierarchy and propagate upwards only as needed (see figure 3.5). This approach harmonizes well

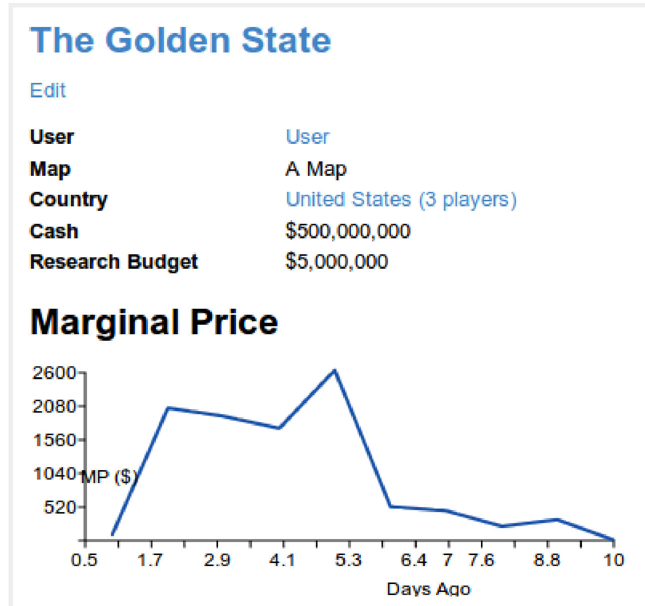

Figure 3.2: A screenshot of marginal price graph. Each State has a graph visualizing the current and historical marginal price of electricity for customers.

with HTTP, which is closely tied to a request/response cycle of communication between client and server. For HTTP, there is no concept of a long-running job that continuously updates the game (see 5.5). Clients should also not have to worry about keeping the game state up-to-date. When a request arrives, the server need only return the best answer it can find, not necessarily the perfect answer. This approach relies on caching at multiple levels of the hierarchy to update the minimum amount of game state necessary to maintain consistency. This frees up processing power for other games, and also increases the response time for players.

3.2.2 Accelerated Game Speed

These update issues are compounded by the fact that games are permitted to scale the passage of time to shorten game duration. Running a power system in real-time speed would be an achingly slow gaming experience, and it would be difficult to observe trends over time. Instead, the time in game can be scaled up as much as 200 times normal. The in-game time is displayed on every page, and begins counting from the *epoch* of the game. This represents an accelerated view into the future, allowing players to see the near- to medium-term implications of their choices.

At the high end of the time scale, update intervals can be especially problematic. At the maximum (200 times real-time), a day passes in game every 12 normal minutes. Nearly every player visit is crossing dozens, if not hundreds of day boundaries (see figure 3.6). The calculations mentioned must efficiently handle updating a large number of days.

3.2.3 `GameTime` Helper

To take advantage of the useful time helper methods in both Ruby and the Ruby on Rails web framework, **threephase** uses a dynamic `GameTime` class when dealing with in-game time. The root problem is that given a `Time` object, the system needs to be able to tell if it has already

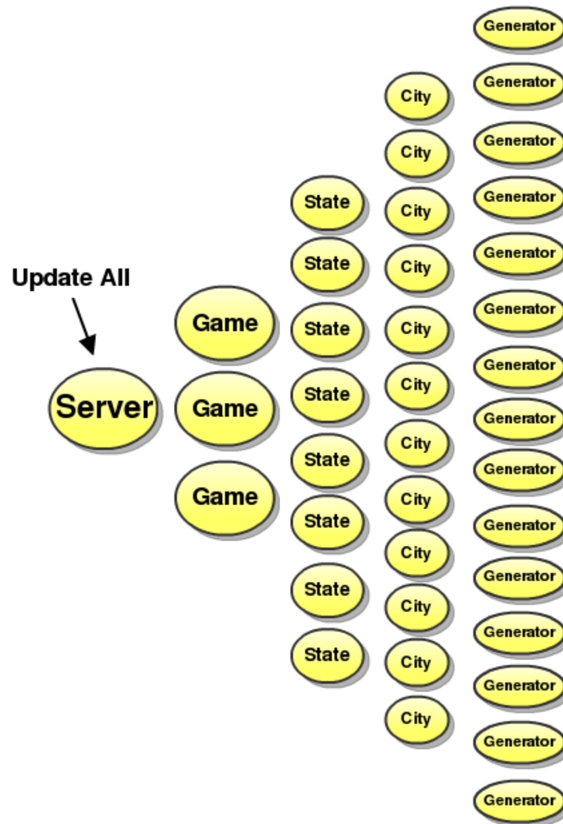

Figure 3.3: A representation of naive, update-all approach to game updating. A naive approach to update scope chooses to update every game object starting and the top of the object hierarchy. This approach is not scalable to many simultaneous game instances, especially with short timesteps in between updates.

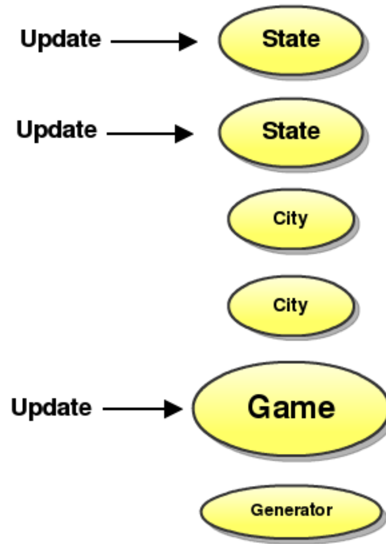

Figure 3.4: A representation of improved, list-based update approach game updating. An improved scope for updates maintains a list of specific objects that need to be updated. It saves time and work over the naive approach, but still has difficulty scaling in small timestep situations.

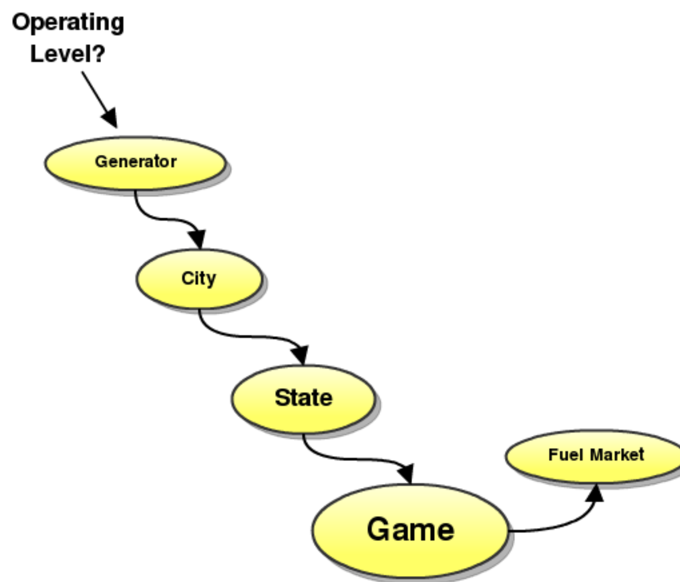

Figure 3.5: A representation of lazy approach to game updating. A lazy approach to update scope updates objects only on demand from player actions. A request for the operating level of a generator may or may not propagate up the hierarchy of objects, depending on the last update time of each object in between.

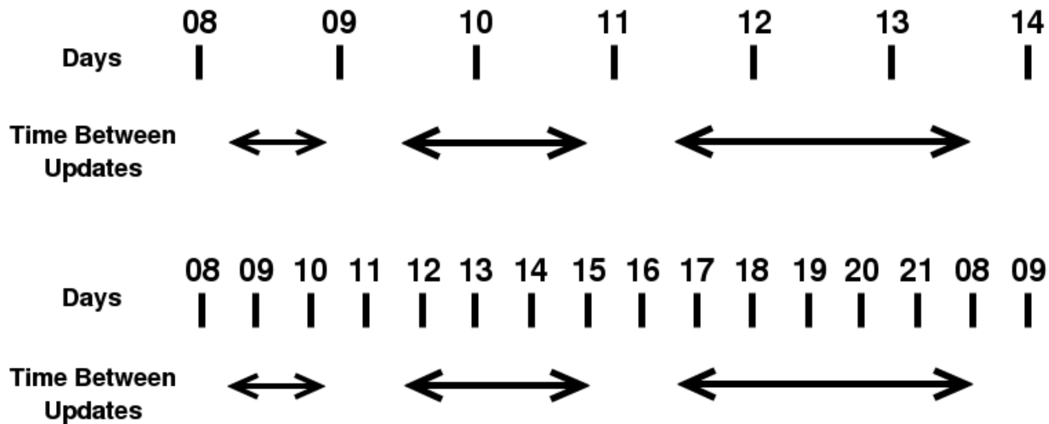

Figure 3.6: A representation of crossing day boundaries between game state updates. Without a steady timestep, all updates must handle the possibility that multiple days have passed since the last update. The lower timeline shows how the day boundary crossing problem is compounded when game speed is increased.

been scaled from real-time to game time. A method receiving an instance of `GameTime` has some implicit metadata (the fact that this is a `GameTime`, not a regular `Time`) that the time is already scaled. In addition, the class provides automatic conversion between real- and game-time as needed.

Each instance of a *Country* generates a unique `GameTime` class definition, with the game's epoch and speed stored as class constants. All times are scaled forward from the epoch (and an error is thrown if a pre-epoch time is passed as an argument), limiting worry about time scaling to a single location in the code base (see figure 3.7).

```
> game.speed
=> 5
> GameTime = game.time
=> GameTime
> GameTime.epoch
=> Sat, 06 Nov 2010 03:38:49 UTC +00:00
> Time.now
=> 2010-11-06 03:39:10 UTC
> GameTime.now
=> Sat, 06 Nov 2010 03:42:08 UTC +00:00
> GameTime.at(Time.now)
=> Sat, 06 Nov 2010 03:42:43 UTC +00:00
> GameTime.at(Time.now).to_normal
=> 2010-11-06 03:39:45 UTC
```

Figure 3.7: Code example of a GameTime scaled time object. The class can smoothly convert between scaled game time and unscaled real time.

Chapter 4

Player Motive

4.1 Demand

In a new state, players will see immediately that their power grid is not meeting customer demand (see 4.1). This is critical in power systems, much more so than any other commodity market. Unlike typical consumer product supply and demand, a lack of supply of electricity doesn't generate consumer buzz like a gadget shortage. The system experiences instability and outages fails if demand and generation don't match exactly.

Notifications Thanks to the way players authenticate with Threephase (with their existing Twitter or Facebook account, using the OAuth protocol), the server can optionally communicate to players out-of-band when such emergency situations arise. Imagine a tweet or Facebook message from **threephase** when generation dips dangerous close to or below the level of demand.

4.1.1 Changing Demand

The first solution to not meeting demand, of course, is to build more generators. Not all is perfect, however, as demand is not static. The load of each *City* (and overall that of the *State*) changes based on the time of day. Each *City* has a predefined load profile function, which determines how that *City*'s demand changes during the day (see figure 4.1). Generally, demand is higher in the afternoon and early evening than late at night. A player's system may be sufficient at 8am, but insufficient later in the day.

Load Profile The current load profile function is static, and simply scales linearly with the number of customers in a *City*. In the future, each *City* could have a more intelligent, varying load profile function. The current function (found by visual approximation) is:

$$-.1 * ((.42 * Hour - 5)^4) + 100) * Customers/Constant)$$

Equipment Failure In addition to changes in demand, components in the system can also fail due to equipment malfunction, natural disasters or union strikes. The system must have enough capacity to withstand the loss of its largest component — this is known as an $n-1$ reliability constraint. As mentioned earlier, the *GeneratorType* determines the relative frequency and severity of failures. The *Country* can also enforce stricter reliability constraints (e.g. $n-2$) for experimentation. The parameters for describing these failures — *technical reliability* (i.e. mean time between failure) and *technical complexity* (i.e. mean time to repair) — are not the

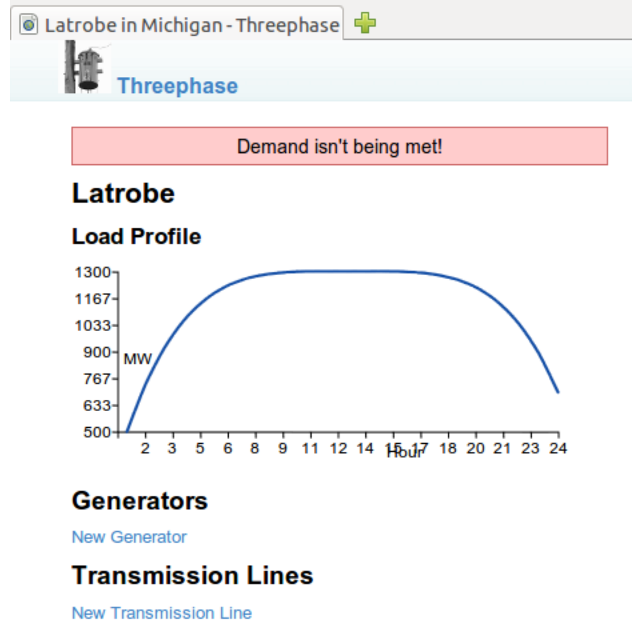

Figure 4.1: A screenshot of a *City*'s load profile graph. Each *City* has an unique load profile function, which determines how demand changes over the course of a day. There must be enough generation to meet the peak demand (towards the afternoon), not just in the middle of the night.

same descriptors used by the electricity industry, but they are more familiar to laypeople and describe similar concepts to the system-wide metrics used by experts (e.g. the system average interruption duration index, or SAIDI). These two attributes determine the frequency at which failures are triggered.

4.2 Primary Player Goals

Players of the game have two basic motives:

- Generate enough power to meet demand
- Route the power generated to the demand

Line Constraints The requirement to transmit power changes the reality of the operating strategy quite a bit. The ideal system, where generators are operated in order of their marginal cost, becomes impossible when the physical location of generators and customers is considered — see figure 4.2 for an example. The line constraint feature was removed from the list of initial features of **threephase**, but it is the next big logical step for simulating reality.

4.2.1 Profit

The true root motive of any utility operator is profit. The ability to make a profit on the system is critical to re-investment in new technology, system upgrades, and in the case of investor-owned utilities, investor satisfaction. This greatly depends on the economic regulatory environment,

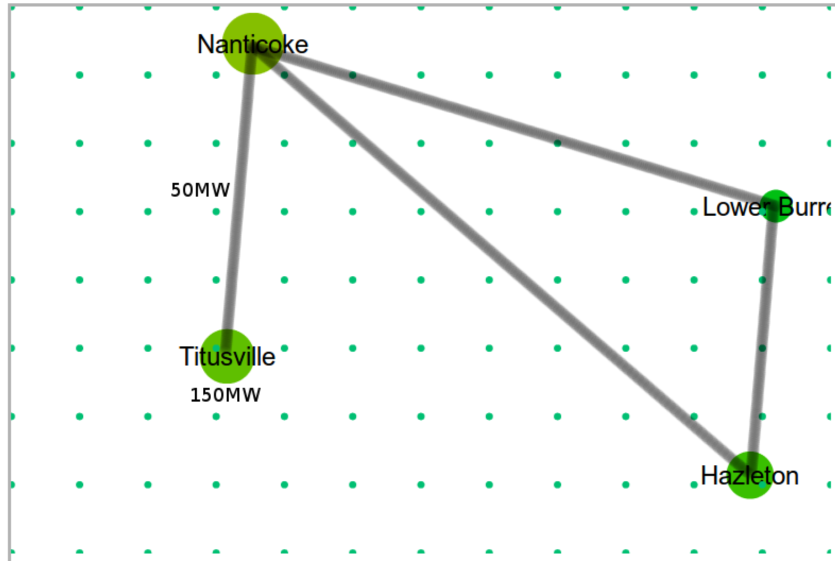

Figure 4.2: A map of cities with physical line constraints. The physical location of load on the grid can make the ideal, cost-minimizing generation scenario impossible. In this example, Titusville has 150MW of demand and no generator, but the only transmission line into the *City* has a maximum capacity of 50MW. It is impossible to transmit enough electricity to meet demand.

both in the real world and **threephase**. The current implementation supports rate of return and marginal cost bidding regulation.

Simplified Operating Costs In all of the regulatory environments, the actual cost of operations depends on the operating levels of each generator. In the current implementation, this is set based on the ideal strategy — generators are enabled in order of their marginal or average cost. Transmission line constraints must be completed before a more realistic scenario can be demonstrated.

Rate of Return

Rate of return regulation is the simplest to calculate and understand. The customers payments are simply the total cost of operating the system at the level demanded multiplied by a regulated rate of return (e.g. 8%). This type of regulation is highly desirable for utilities, as players of the game will quickly realize. A guaranteed return on investment is great encouragement for expanding the system to levels beyond what is actually required. The cost of capital in this system is also lowered, as the risk to banks loaning money is low if the debtor is guaranteed a return on their investment by the government.

In real-world rate of return regulation, there is a possibility that investment decisions made by the utility will not be approved by the regulator. In the future, **threephase** could add intelligence to its regulating algorithm to reject extraneous investment and equipment purchases. In the current implementation, approval is always granted.

Marginal Cost Bidding

The next type of regulation is marginal cost, or average cost bidding. This type calculates the average cost curve each day (recall figure 2.2), and the generators "bid in" at their marginal or average cost (a bid price enforced by the regulators). The market price of electricity for the day is set at the intersection of demand that curve. Generators at the intersection price will break even, generators above it will potentially make a profit and those below are operating below their marginal cost and are thus guaranteed to lose money.

In both rate of return and marginal cost bidding, the generator's operating levels are set automatically each day by the server. An optional operating level override is planned for future versions, to allow players to experiment with and view the effects of market manipulation.

Locational Marginal Pricing

The next regulation type to be implemented in **threephase** will be locational marginal pricing. This regulation type depends on determining generator operating levels that respect transmission line constraints. Each *City* in the *State* (more generally each *node*) has a local price, which is affected by the system-wide transmission capacity, cost of local generation and the local demand. In the example in figure 4.3, Titusville has a marginal price of $80 because of its isolation from high capacity transmission lines, relatively high demand and (not pictured) an expensive local generator to make up the difference.

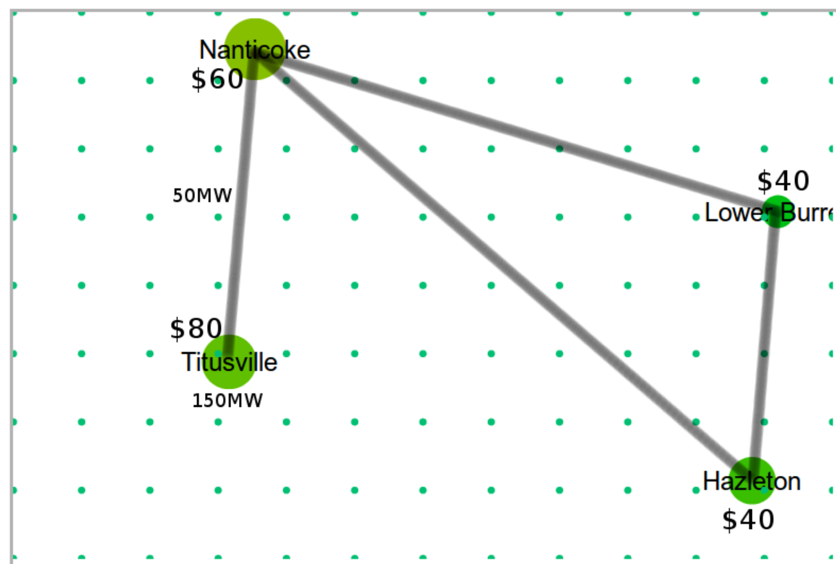

Figure 4.3: A map of cities with Locational Marginal Prices (LMP). LMPs take into account the line constraints when setting the price of electricity. In this example, Titusville pays a higher rate ($80) because of their limited transmission capacity.

Chapter 5

Technical

5.1 Time Distribution

The time spent on **threephase** was tracked over the past two months with the time tracking tool Timetrap [6]. Split into general categories (see figure 5.1), most of the time was spent on the backend code (which determines the data storage and object interaction). The second most time was spent in testing (discussed further in 5.2).

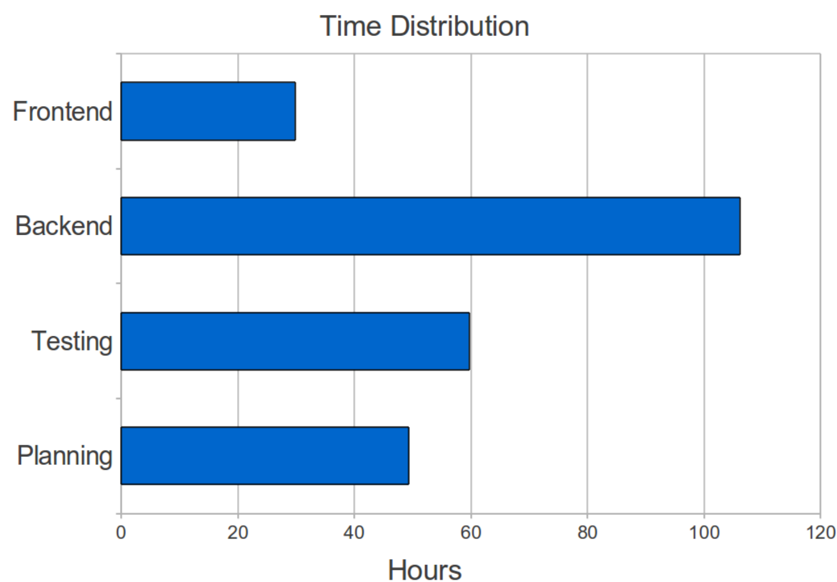

Figure 5.1: Graph comparing the number of hours spent in different areas of the project.

The initial concept of **threephase** evolved over the summer months, culminating in software requirements and architecture documents (now in the project's wiki page). The first month of development was spent implementing the core data models, application controllers and basic web views to interact with the game. The second and final month was spent implementing the game logic — maintaining consistency in the virtual world, updating objects and enforcing the different regulatory conditions.

5.2 Test-driven Development

The development of **threephase** followed the test-driven design philosophy. This method encourages that before any implementation, a test case is written that exercises a small piece of desired functionality. Initially the test will fail since nothing has been implemented. The implementation goal is to write just enough code to pass the test — providing some degree of certainty in correctness and making sure no more code than is necessary is written. The resulting collection of test cases (the *test suite*) is also a critical tool for making sure that contributions from the open source community don't break existing features. Each test case also serves as live, runnable documentation of how a class or method is supposed to work.

Unit Testing At its core, **threephase** is a web application using the Ruby on Rails 3 framework [16]. Most of the interesting logic is in the application's models (i.e. the *State*, *Generator*, etc.), so there is relatively loose coupling to Rails itself. The test suite uses the RSpec testing framework [15] for its human-readable test cases and strong integration with Rails. To test the standard request/response patterns of the game's API, the project uses a custom set of RSpec feature groups. These can be called like methods in a test case to avoid duplication, e.g.:

```
describe StatesController do
  before do
    @game = Factory :game
  end

  context "as an admin" do
    before do
      login_as_admin
    end

    it_should_behave_like "standard GET show"
    it_should_behave_like "standard PUT update"
  end
end
```

Figure 5.2: Code example of an RSpec test case, testing the controller for creating and updating a *State*. This example makes use of the it_should_behave_like feature group ability in RSpec, allowing much of the test logic to be shared among controllers.

Object Factories Instead of test fixtures (preloaded database objects), **threephase** uses the Factory Girl framework to use object factories in test cases. Object factories are simpler to maintain than fixtures, and minimizes work during development to data models that are in flux.

Project Management The project uses the Lighthouse issue tracking system [10] to track milestones and tasks. The initial schedule planned a milestone every four days. This duration ended up being too short, and did not allow for slippage or early finishes. Week long milestones (for a project receiving 3/4 of the time of its participants) would be a more flexible and realistic time period.

5.3 Graphics

threephase originally intended to use the Javascript graphics library Processing.js [14] to render the map, charts and graphs in the browser. Upon further investigation, the library seemed to lack helpful charting features that other libraries offered, and the odd support of syntax from the Java version of Processing made using the language somewhat unnatural (a pure Javascript API to Processing.js was discovered later).

Instead, **threephase** uses two different Javascript graphics libraries: Raphaël [2] for basic charting and mapping and Flot [8] for the piecewise graph necessary for the average cost curve visualization. Raphaël also has an existing charting extension, gRaphaël [1] which reduced the amount of boilerplate graph code that had to be written.

Performance The performance of both libraries is very good in modern browsers (performance tested in Mozilla Firefox and Google Chrome). The bottleneck for rendering complex visualizations at the moment is the time in downloading the data required from the server in the background, not rendering.

5.4 Database

The majority of the objects in **threephase** are stored using the standard object-relational mapping provided by Rails, backed by a PostgreSQL database. The objects in **threephase** (and the physical entities in the real world) are highly relational, so this is a good choice not only for the wide support of SQL databases, but because it fits the data model well.

A few pieces of less relational data are stored separately, in a MongoDB database. This database is well suited to the high insert and read rates required for storing market prices and operating level statistics (these models are also implemented using the standard relational approach from the original draft, as the non-relational version isn't finalized).

5.5 Asynchronous Tasks

The browser-based nature of **threephase** rests on HTTP, the most basic web protocol. HTTP has no knowledge of long running processes, and the relationship between a client and the server is finished after a single request/response cycle. There is not an immediately clear way to mesh this with the very demanding interaction required by games.

Motivation In order to provide reasonable response times, so players don't get impatient waiting for pages to load, the majority of the computation to update the game needs to happen outside of the normal player interaction cycle — whether updating one element or a thousand. A desktop game may use different threads of execution to make sure a player is never waiting for a network packet to finish downloading, or for a texture to load from the hard drive. In web applications, the server can use asynchronous task queues to accomplish the same thing.

Whenever possible, computation is bundled up into a "task" and queued to run at a later time — ideally as soon as possible, so the player gets updated data, but with no guarantee that it will happen before the server returns a response to the client. If the job hasn't completed, it may return cached data that is valid, but not completely up-to-date. There is a trade-off between performance and liveliness, and in this case player perception of the game's speed is

more important than absolutely current information. To accomplish this, **threephase** uses the Resque background job library [21] backed by a Redis database.

Task Examples Examples of work done in tasks include:

- Charging customers based on their demand and the marginal price
- Deducting power grid operating costs over a time period
- Handling random events: research advancements that lower capital costs, unionized worker strikes, etc.
- Clearing the market price of each fuel

Chapter 6

Evaluation

6.1 Criteria

The proposed endpoint of this project was a deployed game server with a playable version of **threephase**. One of the first tasks to make sure this was accomplished was to define the scope of the game for the initial version. Clearly, there is enough material to extend beyond a two month timeframe. The core logic of the virtual world took longer than expected to implement, and as a result progress fell behind the list of planned features. **threephase** is not currently robust enough for a public deployment, and needs additional work to optimize performance and the user interface. The three evaluation criteria from the project proposal were:

- Reception among players. Document reactions during beta testing. *Is the game captivating?*
- Conveyance of critical power systems concepts. *Is the game a useful teaching aid?*
- Robustness and scalability of game architecture. *Is the system well-designed? Are upgrades streamlined?*

The project was not far enough along by the end of the semester to perform user testing, but the introduction of power systems concepts and scalability of the system made good progress.

6.2 Design Changes

threephase evolved from a turn-based game to real-time before the design was finalized. Real-time strategy games are a more natural setting for players, but when the project was proposed the technology for real-time updates in the browser was not mature. While still an emerging technology, recent developments in non-blocking web servers and browser sockets led to a shift in **threephase** 's playing style.

Instead of requiring a somewhat complicated system of action points and player turns, the game world is persistent and never stops. This presented some additional implementation challenges, but in the end will be a more compelling interface.

Chapter 7

Summary & Future Work

7.1 Serious Games

threephase represents a proof of concept of a platform for teaching and experimenting with power system concepts in the context of a familiar game-like interface. The motive roughly follows the ideas behind the Serious Games Initiative (and other, less formal pushes towards games for learning and experimenting). Serious games try to leverage the immersive power of games for education, for both academics and continued learning. These games also serve as excellent training tools within industry. Instead of building strictly academic simulations, experts can make an approachable game, something that laypeople would be interest in playing and the experts themselves would enjoy outside of a class or job.

Player Responsibility Scope A core requirement of serious games is to avoid simplifying critical components, which could alienate the experts of the field, while balancing enough automation that newcomers can learn at their own pace. The game needs to be able to adjust the scope of decisions a player must make, and the extent of the complexity they see, in order to allow players to focus on only certain aspects of the system at one time. Something discovered firsthand over the course of this project is that it can be overwhelming to be responsible for all aspects of the system, coming dangerously close to the micromanagement of resources. The game must balance between system level decisions and real-world issues that high-level simulations often ignore. The player should be able to select from a range of difficulty levels that automate different areas of the system to adjust the difficulty.

Expanded Multiplayer Collaborative learning is also possible with serious games. In **threephase**, each *State* is run by an individual player, but this could be expanded to allow cooperative play — one player controlling the generators and another the transmission lines. This actually reflects another real-world regulatory scenario, where these areas of the power grid are separated by law into different management entities.

7.2 Current Player Actions & Abilities

In the current version of **threephase**, players can:

- Create new games and choose attributes for the game
- Join existing games that have already started

- Build city-local generators from a list of available types and a range of capacities
- View marginal price of electricity over time
- View marginal cost of each generator over time
- View marginal price of each type of fuel

Once the objects are created, the current version of the backend can:

- Calculate market price for each fuel based on supply and demand
- Automatically assign the optimal operating level for each generator
- Order generators based on marginal cost or average cost
- Deduct operating costs (cost of fuel, cost of workforce, etc.) over a time period from a player's cash
- Add customer payments (based on marginal price of electricity) over a time period to a player's cash
  - Calculate marginal price for rate of return regulation
  - Calculate marginal price for marginal cost bidding regulation, assuming a vertical demand curve
- Discount generator operating costs based on map geology
- Trigger random equipment failures (rate determined by generator attributes) - equipment *repair* is notably not yet implemented

7.3 Improvements

There is a long list of features that could be added to **threephase**. The most interesting and pressing items are:

Line Constraints & Location Marginal Prices The implementation of transmission line construction and the respecting of line constraints in determining the operating levels of generators. The game is lacking a key component of real power systems without this. LMP-style regulation depends on this feature.

Avoiding Outages The consequences for not meeting demand in **threephase** are unclear. The player is warned of the condition (see 4.1) and their state essentially freezes in place - customers aren't charged, operating costs aren't deducted, and no power is generated. This is an overly forgiving approach, as there are serious consequences for an outage in the real world ranging from unhappy customers to financial penalties and even eviction from the market. Players in **threephase** are given this great leeway to allow new players a build-up period, where they can build enough generators to meet demand when first joining a game. This phase could be re-worked to occur before players officially join the game and must being running their utility. Once the game has started and harsher consequences are in place, a more useful warning for players will be that "generation is projected to come dangerous close to not meeting generation," thereby giving the player a chance to resolve the situation before an outage.

Load Profiles & Demand Response The load profile of each city is static, and varies only linearly with the population (see 4.1.1). This could be improved not only by introducing more interesting variations, but by incorporating the idea of demand response. If customers are offered time-dependent electricity prices, they (or their networked appliances) can schedule their operating hours to minimize their costs and stabilize the load for generators. For example, electricity is generally less expensive at night due to excess capacity (and can even have a negative price), and customers are unaffected by short voluntary outages of certain appliances. This feature would require additional intelligence in the load profile algorithm, as its value would be based on price as well as time.

Intelligent Map Generation The maps assigned to each *State* must be generated more intelligently, creating a natural a landscape with a relationship between blocks. The current implementation does not allow for realistic groupings of generators around certain resources (e.g. wind farms in a windy area), since the indices can shift dramatically from block to block.

Expanded Multiplayer The multiplayer aspects of the game could be expanded beyond shared national fuel prices to include interstate trade. Interstate transmission lines are implemented but not exposed to the player in the interface. Fuel contracts and contracts for different on transmission have also been proposed.

Interactive Visualizations The map and chart visualizations need to be improved to be more accurate, interactive and useful. The map rendering was intended as the primary interface for the game, but sits as a sidebar in the current implementation. It should present a natural way of viewing the geograph of the *State* and interacting with the generators and transmission lines. **threephase** exposes a JSON API for nearly every function of the game, so the possibilities here depend primarily on user experience decisions.

Mobile Client Because an API already exists, a mobile client would be a good addition to the system, so players can keep track of their power grid without being near a browser.

7.4 Open Source

threephase is intended to be a community project, and has been open-source from the start [18]. The code is hosted on the GitHub code sharing site, along with the project's design documentation. **threephase** will now be reworked from a solo graduate project endeavor to a project shared with people interested in serious games, the power system, web application development, user interfaces and design. The official end of this project also begins its next phase — a call for contributions, ideas and support from the community.

Test Suite The existing test suite will greatly assist in helping contributors add to the system. By running the test suite after making modifications, new developers can make certain they haven't broken existing features. A good test suite can make an open-source project much easier to manage.

Documentation All of the project's documentation is in the wiki pages on GitHub [19]. Where appropriate for developers (and not just a record of design decisions), the documentation will be refactored and moved into the code itself.

Bibliography

[1] Dmitry Baranovskiy. *gRaphaël*. Nov. 10, 2010. URL: http://g.raphaeljs.com.

[2] Dmitry Baranovskiy. *Raphaël*. Nov. 10, 2010. URL: http://raphaeljs.com.

[3] *CityOne*. IBM. Nov. 10, 2010. URL: www.ibm.com/cityone.

[4] *Darwinia Screenshot*. Introversion. Nov. 10, 2010. URL: http://www.introversion.co.uk.

[5] Laurent Fleishman. *Information Materials*. Center for Risk Perception and Communication - Carnegie Mellon University. Nov. 10, 2010. URL: http://sds.hss.cmu.edu/risk/fleishman/InformationMaterials.html.

[6] Sam Goldstein. *Timetrap*. GitHub. Nov. 10, 2010. URL: https://github.com/samg/timetrap.

[7] Jovan Illic. *Gipsys Screenshot*. Carneige Mellon University. Nov. 10, 2010. URL: https://www.ece.cmu.edu/~nsf-education/software.html.

[8] Ole Laursen. *Flot - Attractive Javascript plotting for jQuery*. Nov. 10, 2010. URL: http://code.google.com/p/flot/.

[9] Lester Lave, Marija Illic, and Jovan Illic. *S10 18-875 - The Engineering & Economics of Power Systems*. Carnegie Mellon University, 2010.

[10] *Lighthouse Project for Threephase*. entp. Nov. 10, 2010. URL: http://threephase.lighthouseapp.com/.

[11] *Love Screenshot*. Quelsolaar. Nov. 10, 2010. URL: http://www.quelsolaar.com/love.

[12] Christopher Peplin. *Technical Component Parameters*. threephase wiki. Nov. 10, 2010. URL: https://github.com/peplin/threephase/wiki/Technical-Component-Parameters.

[13] *Power Grid*. Nov. 10, 2010. URL: http://www.riograndegames.com/games.html?id=5.

[14] John Resig. *Processing.js*. Nov. 10, 2010. URL: http://processingjs.org.

[15] *Rspec*. Nov. 10, 2010. URL: http://rspec.info/.

[16] *Ruby on Rails*. Nov. 10, 2010. URL: http://rubyonrails.org/.

[17] *SimCity 2000 Screenshot*. The Game Oldies. Nov. 10, 2010. URL: www.gamegoldies.org/simcity-2000.

[18] *Threephase GitHub Repository*. GitHub. Nov. 10, 2010. URL: https://github.com/peplin/threephase.

[19] *Threephase GitHub Wiki*. GitHub. Nov. 10, 2010. URL: https://github.com/peplin/threephase/wiki.

[20] *U.S. Energy Information Administrator*. U.S. Department of Energy. Nov. 10, 2010. URL: http://www.eia.doe.gov/.

[21] Chris Wanstrath. *Resque*. GitHub. Nov. 10, 2010. URL: https://github.com/defunkt/resque.

[22] *Zynga*. Nov. 10, 2010. URL: http://www.zynga.com/.