COMP 400 Report

Balance Modelling and Analysis of Modern Computer Games

Shuo Xu

School of Computer Science
McGill University

Supervised by Professor **Clark Verbrugge**

April 7, 2011

Abstract

As a popular topic these days, balance plays an important role in modern computer game design. Game producers often need to configure balance of their game systems so as to provide players with a fair and enjoyable game environment. Although the industry has put much effort on this issue, few theoretical analyses have been setup in real life. In this paper, we build an analysis model based on PVE systems, to investigate several balance related problems, especially the complexity problem. And we work out a generic metric to measure the balance of state-of-the-art commercial games.

1. Introduction

Game Balance is probably one of the most complicated and hardest problems in modern computer game data and content design. Up to now there is no proven way to achieve perfect balance. The balance issue appears in many different genres of games, and spreads out among different parts of the same game. It can be mainly separated into two domains: PVP based and PVE based. By focusing on the latter one, in this paper, we propose a generic method to model the balance system in a PVE based environment.

2. Game Balance

Balance issues appear in several different manners in modern games. As we classify them into two basic types – PVE and PVP, we investigate their objectives, structure and difficulty respectively.

2.1. PVE based balance

Player Versus Environment is a straight forward type of game play. Key concepts include interaction between the player and the game world itself (environment). The balance issue is often concerned in two directions.

One is the consistency of various action sequences towards the game goal. For many games, multiple ways to achieve the target (win-state) are usually implemented. For a standard MMORPG(Massively Multiplayer Online Role Playing Game), levelling up procedure usually provides different ways for player to choose, either by doing quests in map A, or by keeping grinding (killing monsters for experience) in map B, etc. The game world is aimed to be designed in such a way that whichever strategy a player choose, it should not vary too much in terms of time consumption, and gaining rewards.

The second direction is the playability and fun. An over-challenging game or ridiculously easy game is obviously not acceptable for audience. A proper way to balance the difficulty of environment, creatures, as well as choice branching factors (to avoid getting overwhelmed in game) is the key to fix this problem.

2.2. PVP based balance

Player Versus Player is a totally different story. It handles the interaction between human players, and is usually implemented in a non-state based, real time environment. Concerning the balance in PVP, the designer often wants to provide the game with a fair competition basis for the players using different classes, characters, roles, etc. Examples include MMORPG class/faction balancing, RTS (Real Time Strategy) race balancing, FPS (First Person Shooting) weapon and map balancing, and so on.

The difficulty of analyzing PVP balance lies on the continuous environment we are facing. Actions can be made in any time slot, which implies an infinite amount of possible game states. Hence in this work, PVP related topics will not be heavily discussed.

3. Analysis Model

PVE games themselves usually contain several systems that are concerned with balancing.
Here in this research, we select the Quest System from a commercial MMORPG called:

"PERFECT WORLD INTERNATIONAL" (2008) [1]

by Perfect World Entertainment Inc. as our base model.

3.1. Quest System and Balance

In the procedure of levelling up in a traditional MMORPG game, **quest** plays an important role for its functionality of providing rewards, experience to make progress, and background information about the game. Generally quests are associated to each **level** of a game, where the “level” serves as a standard for evaluating current player's standing or status in game community. A high level for a character usually means stronger abilities and skills in PVE and PVP, as well as higher reputation. *(Notice: “character” and “player” are used exchangeably in this paper.)* For each level that a player reaches, a set of quests will be automatically activated by the system. It is player's decision to choose which quest to do and in which order when they become available. Doing quests is essential in our game Perfect World International as it is the main way of upgrading levels.

A quest can have many attributes attached to it, typically those include:

- Start NPC (Location), Completion NPC (Location)
- Detailed mission, Rewards (Gold, Experience, etc.),
- Difficulty (Complexity, Time needed),
- Restrictions (Dependencies with other quests (e.g. Pre-Quest)

By doing quests in different order, the player would get different game experience in terms of rewards earning and costs of time. This is because of two reasons.

One is **location**. Doing quest A would ends up with a location of F(xA, yA) (finish location for the quest A) for the character. Once player wants to do quest B next, it needs to approach to the Start Location of B, namely S(x_B,y_B), from F(x_A, y_A). We calculate and set this distance of moving as D(A,B).

However, if we reverse the order of quest accomplishment, we will get D(B,A) , which is the distance of S(x_A,y_A) from F(x_B,y_B). This is most probably different from D(A,B). The following **Figure 3.1** explains the idea more intuitively:

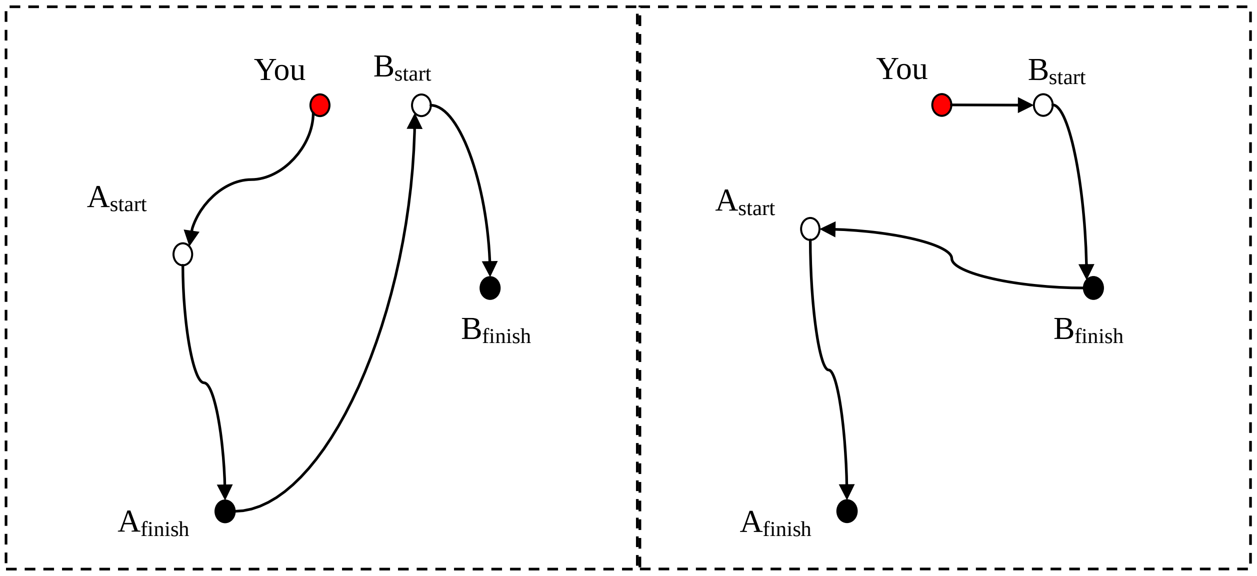


Figure 3.1

Different distances means different time costs, where we have the same rewards at the end interestingly. In both case we finished Quest A and Quest B, whose rewards R(total) = R(A) + R(B).

The second reason is **dependency**. Assume Quest D can be started only if the player has completed Quest C. (Quest C is the Pre-Quest of Quest D). Another Quest E has no relations with others. Now choosing Quest C or Quest E at the first place will result in different amount of choices left:

Doing C – We can do D afterwards, and we might be able to level up without doing E.

Doing E – We still need to do C if D's experience is unfortunately needed for reach next level.

Depends on the choice player makes, he or she will end up with different time cost and probably different rewards gained.

With these concerns, **balance** comes into play. Since players would have various outcomes by their decisions, we need to ensure that all those decision will not be too much different from each other, so as to maintain the fairness and consistency of the game.

3.2. Brief Summary of Objectives

Player wants to upgrade its *Levels*. Levelling up requires experience from completing *Quests*. By doing quests in different order, we may have different *Costs*. (As explained in Section 3.1)

Avoiding **shortcuts** of winning in such a procedure (Less time to level up very quickly) leads to our balance issue, which will be discussed throughout this paper.

3.3. Quest Tree

In order to extract all the balance-related information we need from the game quest system, we decide to build a game tree that expresses both the logic sequence (quest dependencies) and the information of the states we arrive at.

Each node represents a **state** of the character. A state contains:

- StateID = hashing value of a state, also used as index
- gold = gold obtained so far
- exp = experience points obtained so far
- spirit = spirit obtained so far
- time = total time used so far
- location = current location of the player
- dis = total distance travelled so far
- questDone = list of ordered quests that are completed

Each edge represents an **action**, in our case, the single Quest.

A simple example of Quests from Level 1 is shown in **Figure 3.3,** note that it is a simplified version just to illustrate concept.

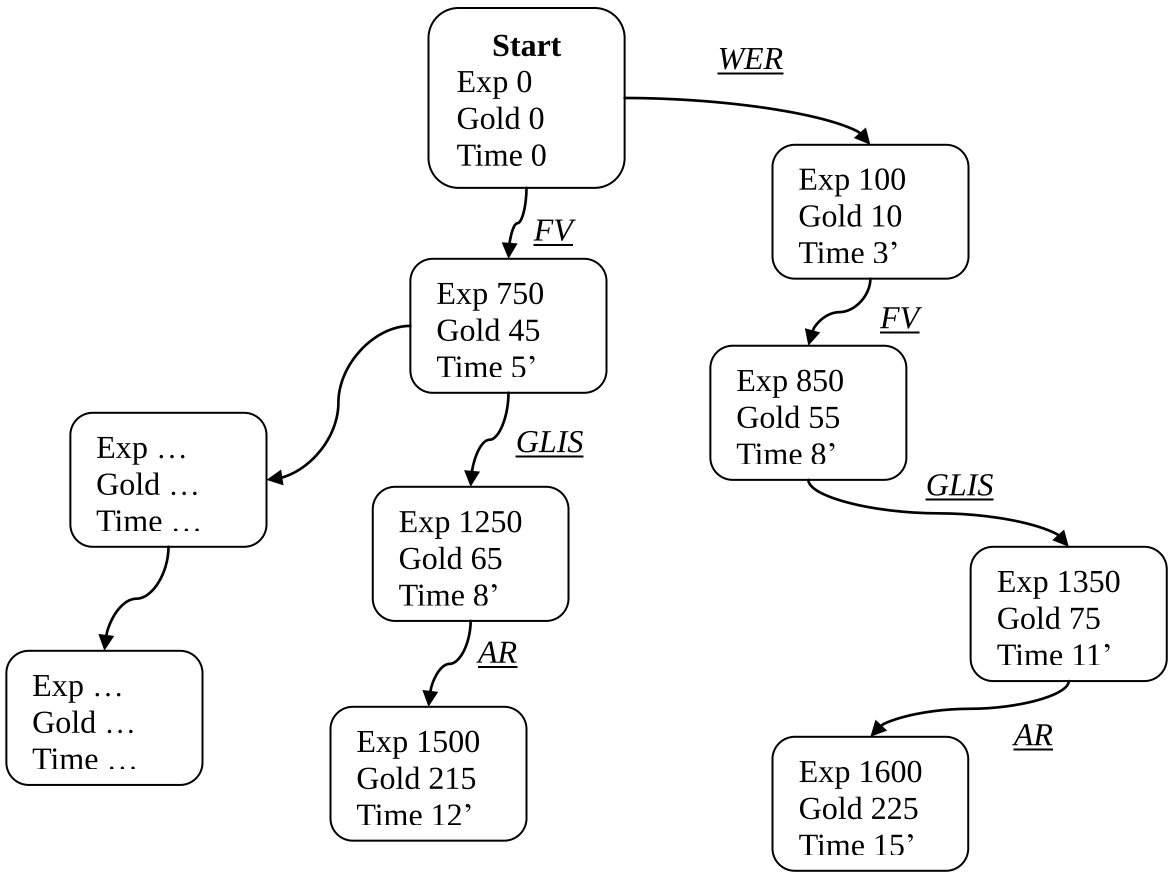


Figure 3.3

4. Single-Level Approach

The main idea of single level analysis is to evaluate the quest tree in a level-by-level manner, without concerning about the cross-level communications of quests. Each quest is attempted as long as it becomes available to the player. This is the **local approach**.

4.1. Single-Level Structure

For each level of the game, we generate the corresponding search tree, and produce all the possible game states that can be achieved only in that level.

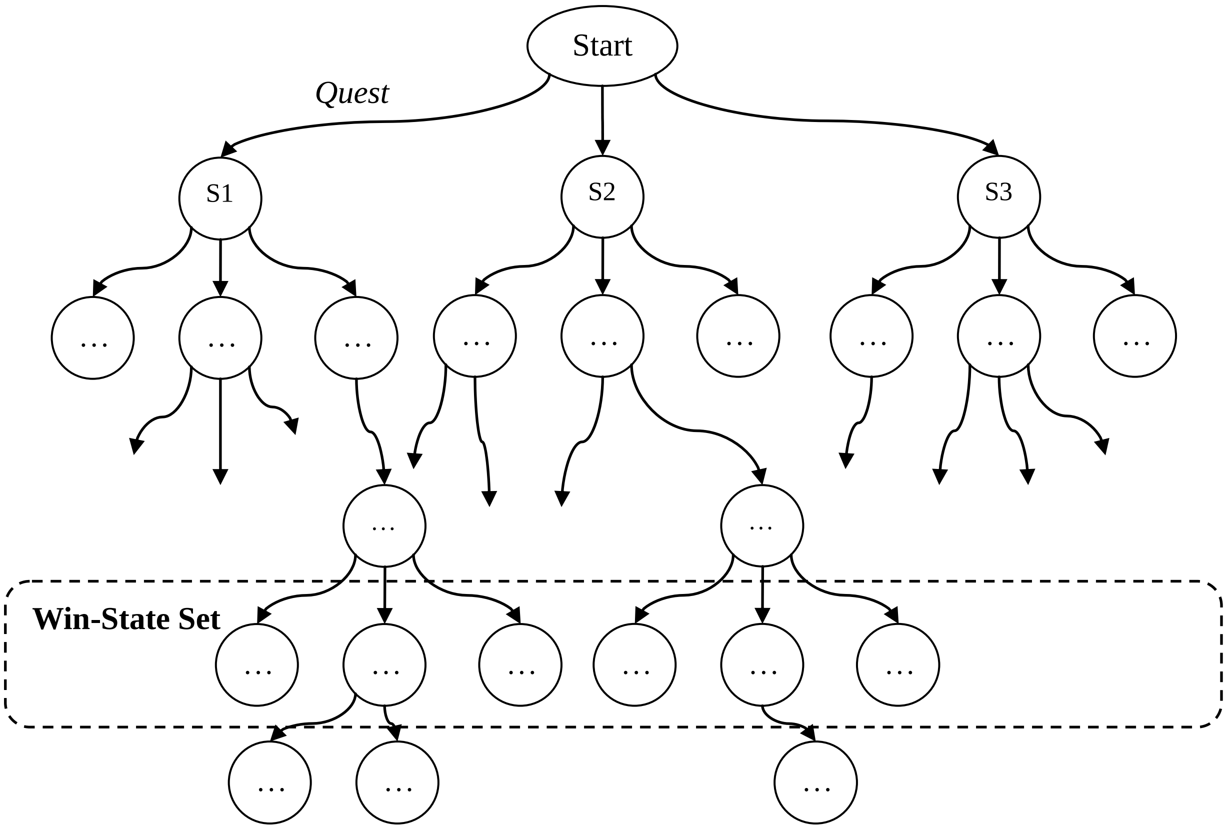


Figure 4.1

Figure 4.1 shows a typical quest tree for level X. After traversing every possible quest that is available at the start of level X, we end up having a group of nodes (states) surrounded with dash lines. We call it the **Win-State Set**, inside of which each state has fulfilled the requirement to leave the current level. They are not necessarily the leafs that stay at the bottom of the search tree, but rather the end points that a player can choose to step into next.

4.2. Complexity

The complexity of the single level approach is, unfortunately, exponential. For each layer of the tree, we decrement the size of the available quest set by 1, which equals the branching factor for the next layer. This gives us a factorial number N! for the total amount of nodes at the bottom that is very close to the size of the Win-State Set.

Due to the small number of quests for each level of our specific game Perfect World International – a number around 6, we are able to generate the whole tree in few seconds. However it is not applicable to more complex game such as World of Warcraft, where each level can contains over 30 quests. Formation of the search tree would take thousands of years.

This problem is not unsolvable by applying pruning strategy, as will be discussed in the following sections.

4.3. Limitations

Besides the performance issue of single-level analysis, there are also many other limitations that are critical.

First of all, it is not always a good idea to try and complete each quest in a certain level. In practice most human players choose to do part of quests that seem convenient to them, instead of exploring everything. Therefore, the basic approach is too naive in this sense.

Second, analyzing the quest system level by level is not representative. As we only generate the data for a particular level, we cannot obtain any global information about the game. Local balanced quests do not mean the whole game is well balanced.

Furthermore, since the basic approach does not provide interactions with other parts of the game, we cannot observe the trend of the game design. Usually an MMORPG game can have different targets on players at different time, such as the **exploration phase** at the start of the game when designers want the player to fully explore the world (in order to quickly get involved in their games), or the **challenge phase** at high levels when the players are desired to compete with each other (PVP) rather than spend their time working on quests. The basic model is not able to provide any information about this rationale.

For the above concerns, we move on to the analysis model where several levels are traversed together.

5. Multiple-Level Approach

Due to the limitations of basic level approach, we decide to model the quest system multiple levels at a time. In multiple-level approach, we select a range of levels instead of one individual, and build the search tree for all the quests involved.

Sadly, as mentioned in the single-level model, complexity problem becomes even worse here in this multiple-level manner. Since now we are analyzing a huge pool of quests from several levels, the Win-State Set of this model grows up quickly to over 20! The search algorithm of finding those end states is no longer efficient. For solving this obvious but essential problem, we come up with the idea of using divide and conquer method – **"Train" Structure**, combined with the **Similarity Pruning** and **Sampling**.

5.1. "Train" Structure

In order to limit the quest tree depth of multiple levels, we group the quests according to their levels. To do so, we apply the basic single-level analysis for each level in our domain. We no longer need to deal with a long quest path anymore, but simply search each local tree instead.

After generating the local quest tree that only contains quests for that particular level, we mark the local Win-State set an **End Set**. This End Set will be transformed into the start node of the next level by transformation function **α**.

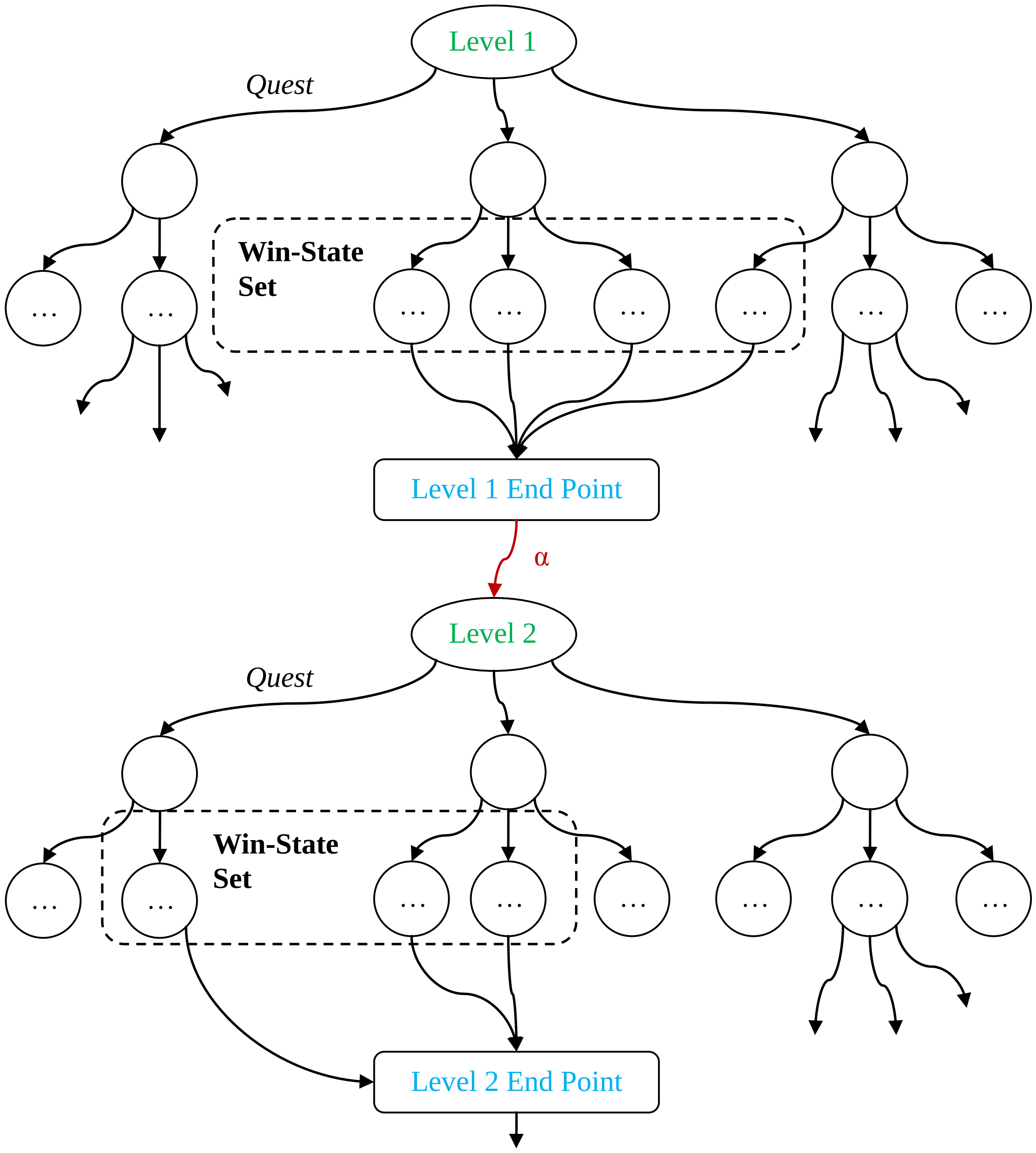


Figure 5.1

Figure 5.1 Shows the structure of an example Quests "Train". At each connection between two levels, we narrow the size of leafs by just keeping Win-State Set, thus control the size of the whole multi-level search tree. (We drop the nodes that are not contained in the Win-State set)

Now a critical problem occurs. How do we transform a set of nodes, namely the End Set of last level, into an individual start-state node for the next level?

Since each node represents a unique state, we cannot simply drop off nodes from End Set and just keep one as a “good start” for continuing. We need to design a nice transformation function **α**. Here we decide to treat all these nodes equally, so that each of them is the start nodes for the next level. This means we have a group of start nodes instead of one. The obvious trade-off is the complexity. By implementing the “Train” structure in this way, even we have already drop the “useless”(not in Win-State set) leafs, we still end up with exponentially increasing size of start nodes, and finally we can be out of control.

To solve this issue, we come to the Similarity Pruning and Sampling Strategy.

Important Note: The purpose of applying Pruning and Sampling in our transformation ***α*** *is to reduce the size of End Set into* ***non-exponential*** *range, with a tolerable accuracy (quality) loss.*

5.2. Similarity Pruning

The idea of **Similarity Pruning** is to compare all the nodes in the End Set, remove the ones that are *similar* to someone else, and turn the result set into the **Start Set** of next level. For example, End Set can be [1.0, 2.9, 1.2, 3.0, 0.8]. After similarity pruning, the result is simply [1.0, 2.9], while other values are similar to these two.

However, how to define “similar” for our nodes?
In our case, each node represents a single state achieved by the player, with at least *six* attributes associated to it. This means we have to perform six degree comparisons. Intuitively, we compare each of the attributes between nodes, if they are different, we continue to the next one until we find a similar one. We remove the similar one so as to decrease the size of the End Set.

In practice, we find out that applying this intuitive method we would end up with the End Set that is not pruned at all. It is very unlikely that two nodes would have all the six same attributes among this big search tree. Therefore, in order to solve this problem, we introduce a **tolerance T**, which indicates the largest variance allowed in each attributes of our nodes. The following example of 3 nodes (states) shows what exactly does tolerance T mean:

A: [Gold-10000, Exp-5000, Spirit-2000, Time-30 minutes, Distance-100 units, 5-Finished Quests]
B: [Gold-11000, Exp-4800, Spirit-2100, Time-28 minutes, Distance-105 units, 5-Finished Quests]
C: [Gold-17000, Exp-5100, Spirit-1500, Time-31 minutes, Distance-100 units, 5-Finished Quests]
T = 0.2

Here we say *state A is similar to B*, since for each attributes of both states the variance is smaller than T.
And *state A is different from C*, because one of the attributes Gold has variance:

(Gold (A) – Gold (C)) / Gold (A) = 0.7, greater than the tolerance of 0.2.

After several tests, we finalize the value of T to be **0.15**, with which the new pruned End Set is largely decreased in size yet maintain a high quality of data for further analysis.
Note: high quality means the representative nodes (states of game) are kept in the set after pruning.

5.3. Sampling

Although by using Pruning Strategy we narrow the “width” of quest tree tremendously, we still have not yet escaped from the exponential growth. Say originally we have 680, 730, and 560 states for the End Set (local Win-State Set) of the single level: Level 10, Level 11, and Level 12. After Similarity Pruning we might reduce these numbers to 370, 420, and 290 for Start Set of Level 10, 11, 12 respectively. We still have exponential growth if we connect these 3 levels into a “Train”. The final result set from these 3 levels would have a size of 370*420*290, and would go out of control if we add more levels for analysis.

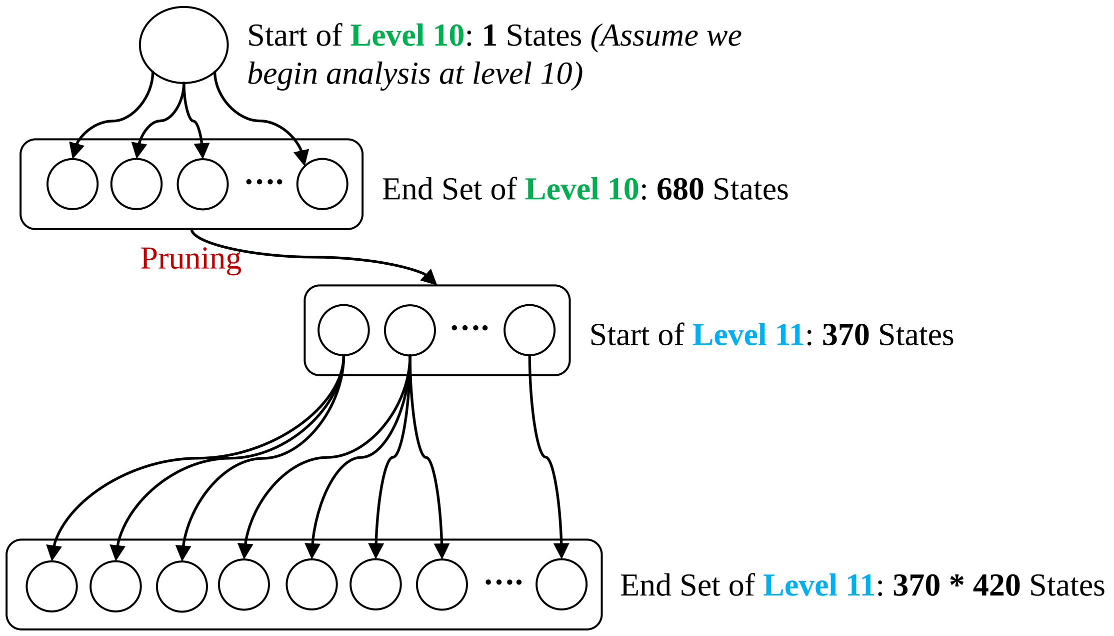

Start of Level 10: 1 States (Assume we begin analysis at level 10)
End Set of Level 10: 680 States
Pruning
Start of Level 11: 370 States
End Set of Level 11: 370 * 420 States

Figure 5.3.1

Figure 5.3.1 illustrate why we still get exponential growth by only applying Similarity Pruning

To completely reduce the complexity to non-exponential as desired at the beginning, we resort to **Sampling**.

Sampling is a simple idea by just choose a **fixed** number of nodes from the Start Set of each level, and discard all the rest. Obviously as a consequence, at the start of each level, we can ensure that only a limited number of nodes are left, regardless of how big the original pruned End Set is. We just always keep, say, 50 nodes at each level. The concept can be viewed in **Figure 5.3.2**.

As you may have already considered, a very big overhead here is the accuracy of data. How we can drop over 90% of the nodes in our result without losing any quality? So we need to decide carefully on the way we select these fixed number of samples.

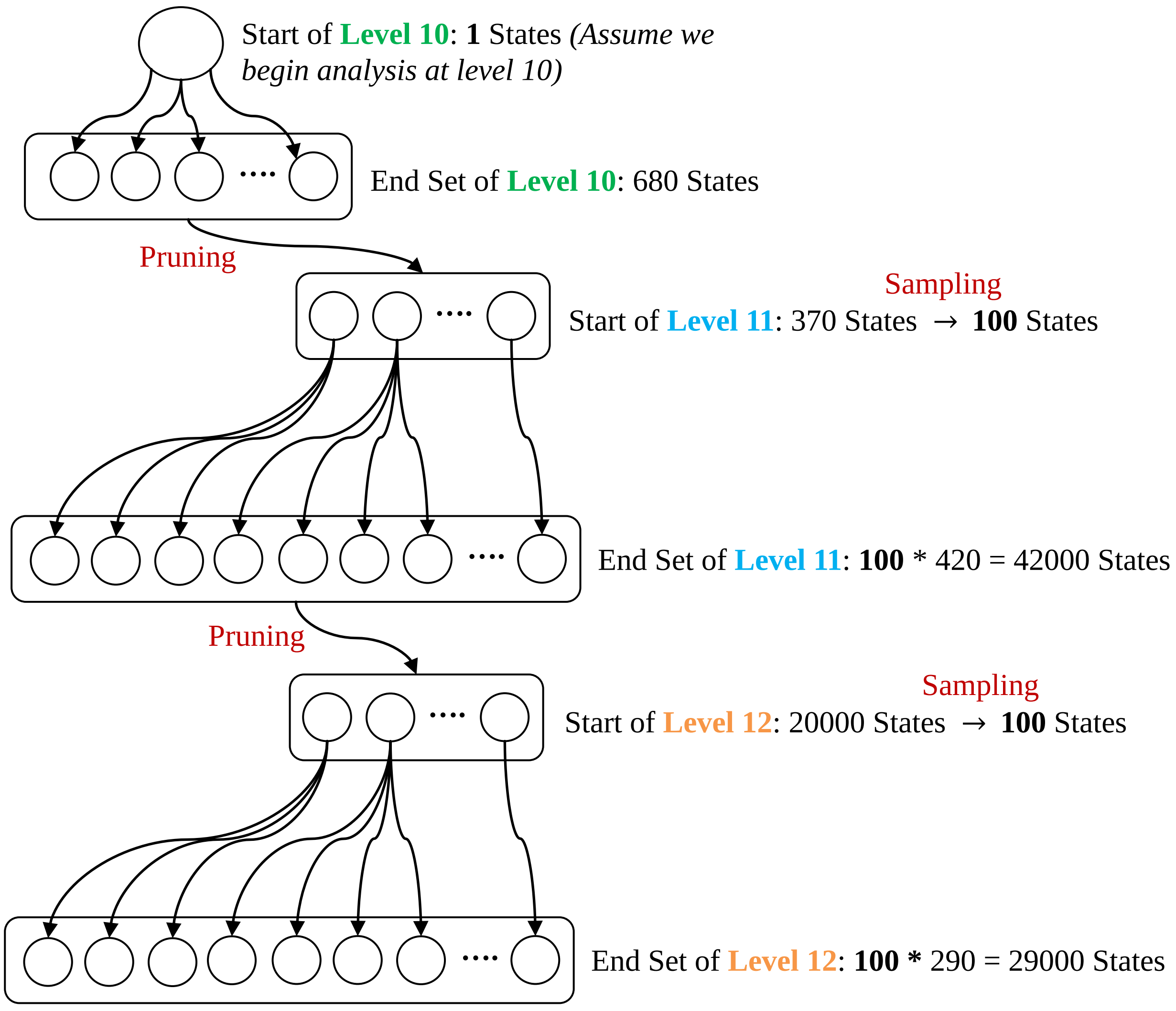
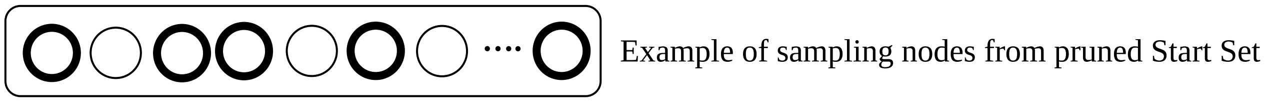

Start of Level 10: 1 States (Assume we begin analysis at level 10)
End Set of Level 10: 680 States
Pruning
Sampling
Start of Level 11: 370 States → 100 States
End Set of Level 11: 100 * 420 = 42000 States
Pruning
Sampling
Start of Level 12: 20000 States → 100 States
End Set of Level 12: 100 * 290 = 29000 States

Figure 5.3.2 Basic idea of Sampling

First we should keep the **extreme nodes** (node with largest sum of all attributes and the one with smallest sum) in our samples, so that the bound of the set remains unchanged. For the rest of samples, we apply **randomized** choosing on all the nodes left. This is a generic method of selecting samples without knowing the actual weight (how important) of each attributes in every state. So we just keep this strategy for the possible future works.



5.4. Summary of Multiple-level Analysis Model

By using Similarity Pruning and Sampling method in our "Train" structure, our quest system analysis model becomes stable and accurate enough for the real experiment. As we process multiple levels at a time, the model is also representative of the game from global point of view. A snapshot of the model can be seen in **Figure 5.3.2**

We will use this as our base for the following experiments.

6. Experiments and Output Analysis

6.1. The Program

For the experiment, we build our program by using Java SE6.0 and Eclipse Environment. As can be seen in **Figure 6.1**, the main program lies in Analyzer, and it will invoke both BasicLevel and MultiLevels, who would return the output of our analysis and meanwhile create the file containing all the data produced. The original level information of the game **PERFECT WORLD INTERNATIONAL** is downloaded from its official website and is imported to our program inside BasicLevel.

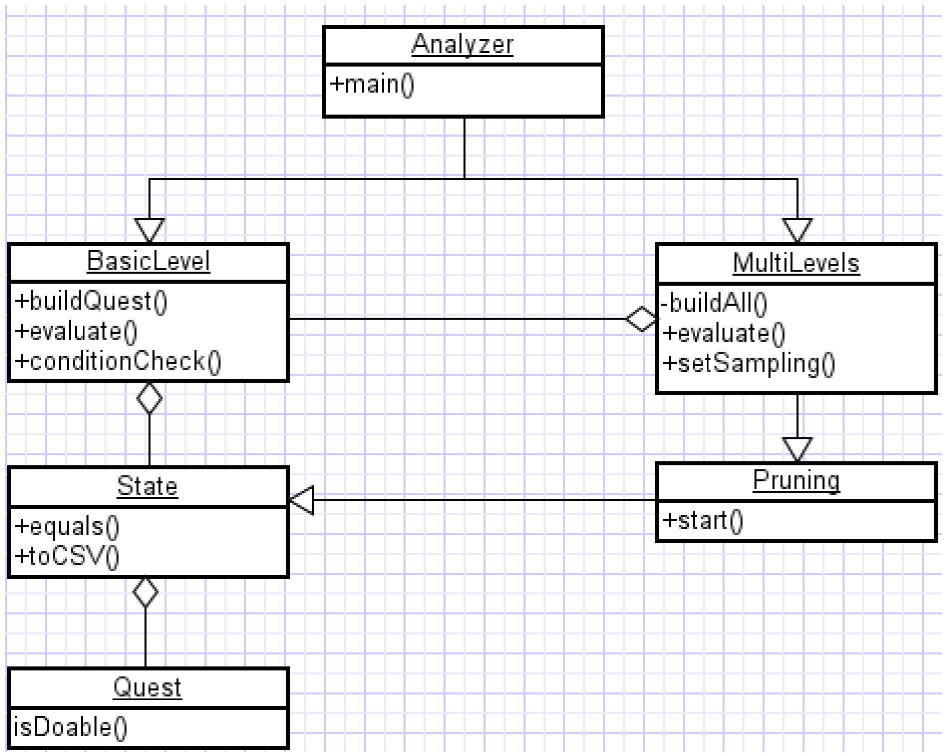


Figure 6.1 Class Diagram of our Program

6.2. Experiment 1 (61-70)

This experiment contains analysing 10 levels from **Level 61 to Level 70**. These 10 levels represents the important period of the game (105 levels totally) where balance issues are concerned most. In our experiment, we only keep the most representative and essential attributes, namely, Time, Experience, and Gold.

The output files are created as *400data_61_to_70_full.csv* and *400data_61_to_70_plot.csv* in the result_lvl_61_70 directory under the program package. Now we use the data in *_plot.csv* to draw the graph by GnuMetric Software, so as to illustrate the output more clearly and reading-friendly.

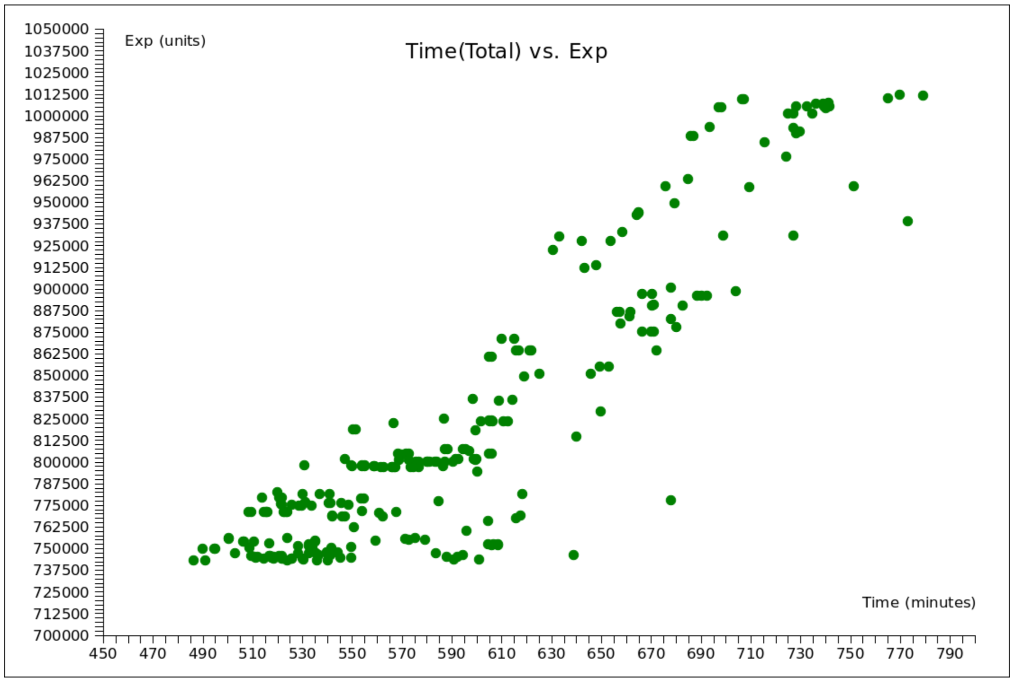


Figure 6.2.1

Figure 6.2.1 shows the relation between Time and Experience of the game, probably the most important relation concerning the balance issue. In the graph, each point represents an individual state of the game. This point (state) contains the completed-quests vector so that we can trace back all the quests and information to get to this state.

From the graph we observe an almost linear distribution. This distribution shows the local trend for the 10 levels from 61 to 70. It implies that the general balance design is good at these 10 levels.

However, several noisy points can be seen at (680, 775000), (640, 737500) and (775, 940000). These three exceptions indicate that the sequences of quests leading to those states are not well designed. In other word, we spend more time (Time) of game play than usual, but the reward (Exp) is not large enough compared to other points. These exceptions are tolerable since the proportion is low, 3 out of 253 samples.

Another phenomenon that can be observed from this graph is that there seems to be more points at the low region (bottom-left part). This might be a result from so-called “Exploratory Period”. In the exploration time of these 10 levels, often game designers want to make the quests easy and short, so that players can explore and understand their game world quickly and in a comfortable way. As player gets more experienced in game, a few choices of challenging themselves for higher reward appears, as shown in the high region (top-right part of graph). That is why we see the clustering of points in the graph.

Experience is not the only attributes that is contained in the game system. Let's take a look at how other attributes behaves.

Figure 6.2.2 describes the relation between Time and another reward option – Gold. Comparing it with the graph of Time vs. Exp, we can see an obvious less dense pattern. All the points seem to be more or less apart from the central linear line.

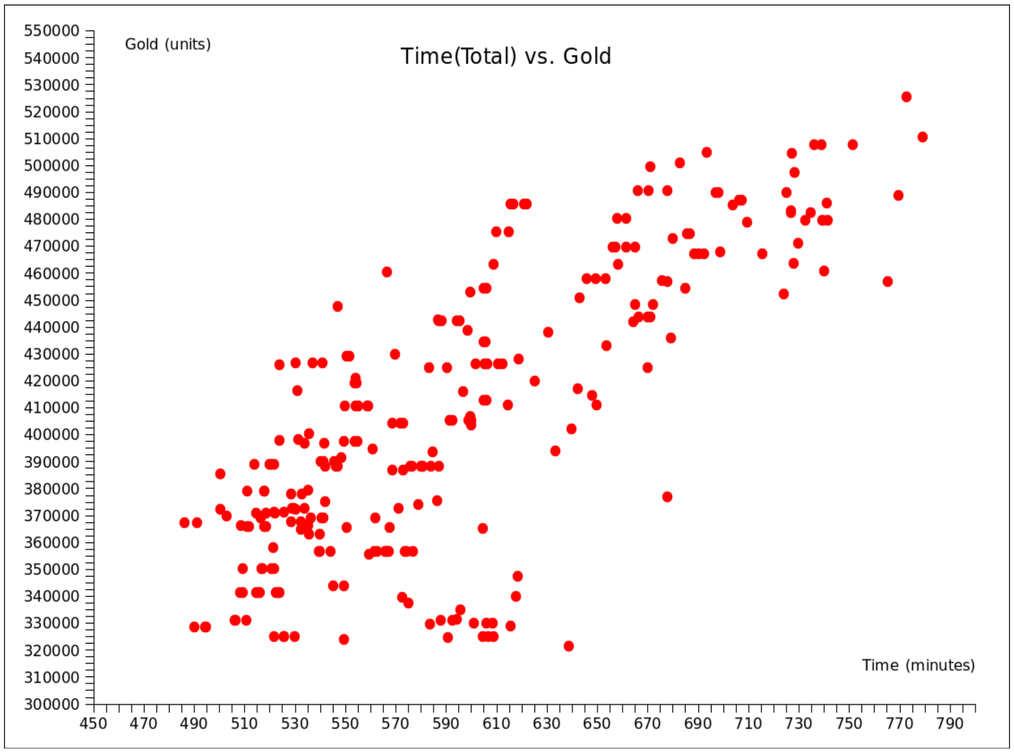


Figure 6.2.2

The reason for this loose distribution is that Gold is not as important as Experience in the game design. **PERFECT WORLD INTERNATIONAL** is a game whose game play balance mainly focuses on character's levels and the corresponding experience points. So the weight of experience is greater than that of the gold in this game.

Interestingly, we can still observe the similar clustering as in previous Exp graph, which means the classification of exploratory period and the advanced period also exists here.

After investigating the Time-related balance, we produce another graph that explains the relation of Gold and Experience.

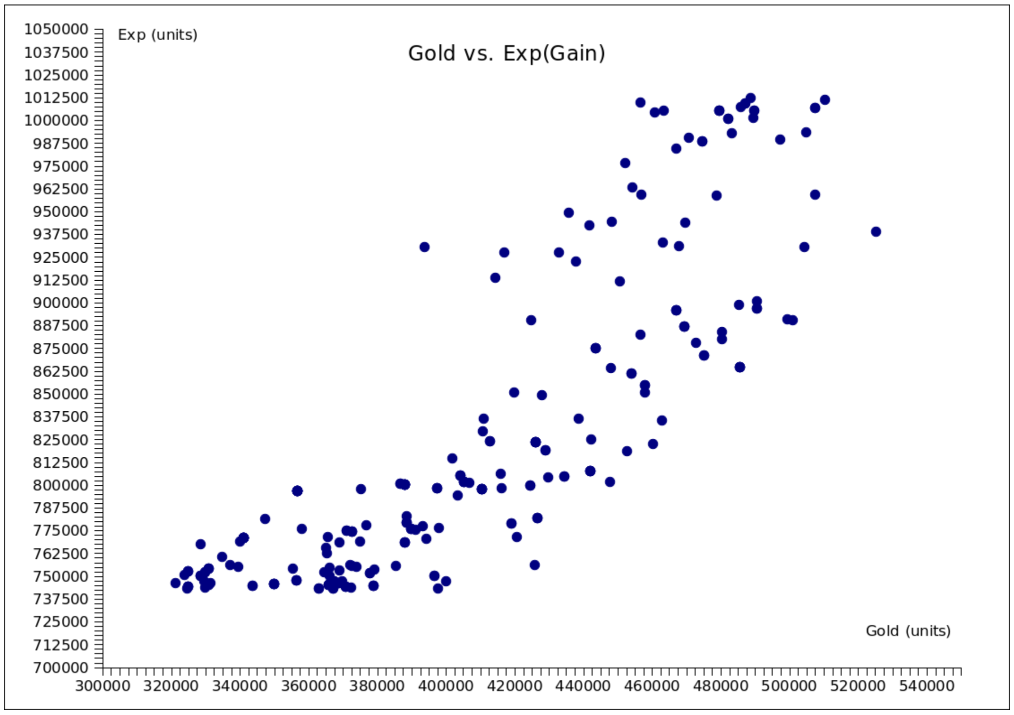


Figure 6.2.3

From **Figure 6.2.3** we can see that the correlation between Gold and Experience is still loose in that many points are far away from the center line. However this is what we have expected. The gold in our game can be affected by many factors, not only the quest system, but also the other systems such as trading, grinding and Team Raids. Weak correlation of Gold and Exp does not necessary means bad balance design of these attributes values, unless we include every piece of the game into our analysis model.

Having looked at these relations from Level 61 to Level 70, we now have a brief understand on how data interact with each other and how balance lies between them. In order to further confirm our result, we approach to the next experiment, which deals the quest system at level 31 to 40.

6.3. Experiment 2 (31-40)

Level 31 to Level 40 is considered the exploratory phase of the whole game, instead of the more advanced phase in Level 61 to Level 70.

Using the same model and method, we are able to generate the following result in **Figure 6.3.1**.

It is not difficult to see that for Level 31 to 40, the pattern of all three graphs is similar to that of the advanced phase starting from Level 61, except that more points are clustering under the low bottom-left region. This is due to the exploratory characteristics of this whole period in the global game range. Certainly at the early level 31 or 32, designers would put more effort on making the game easy to play, and less time-consuming in order to keep players attraction for their commercial product.

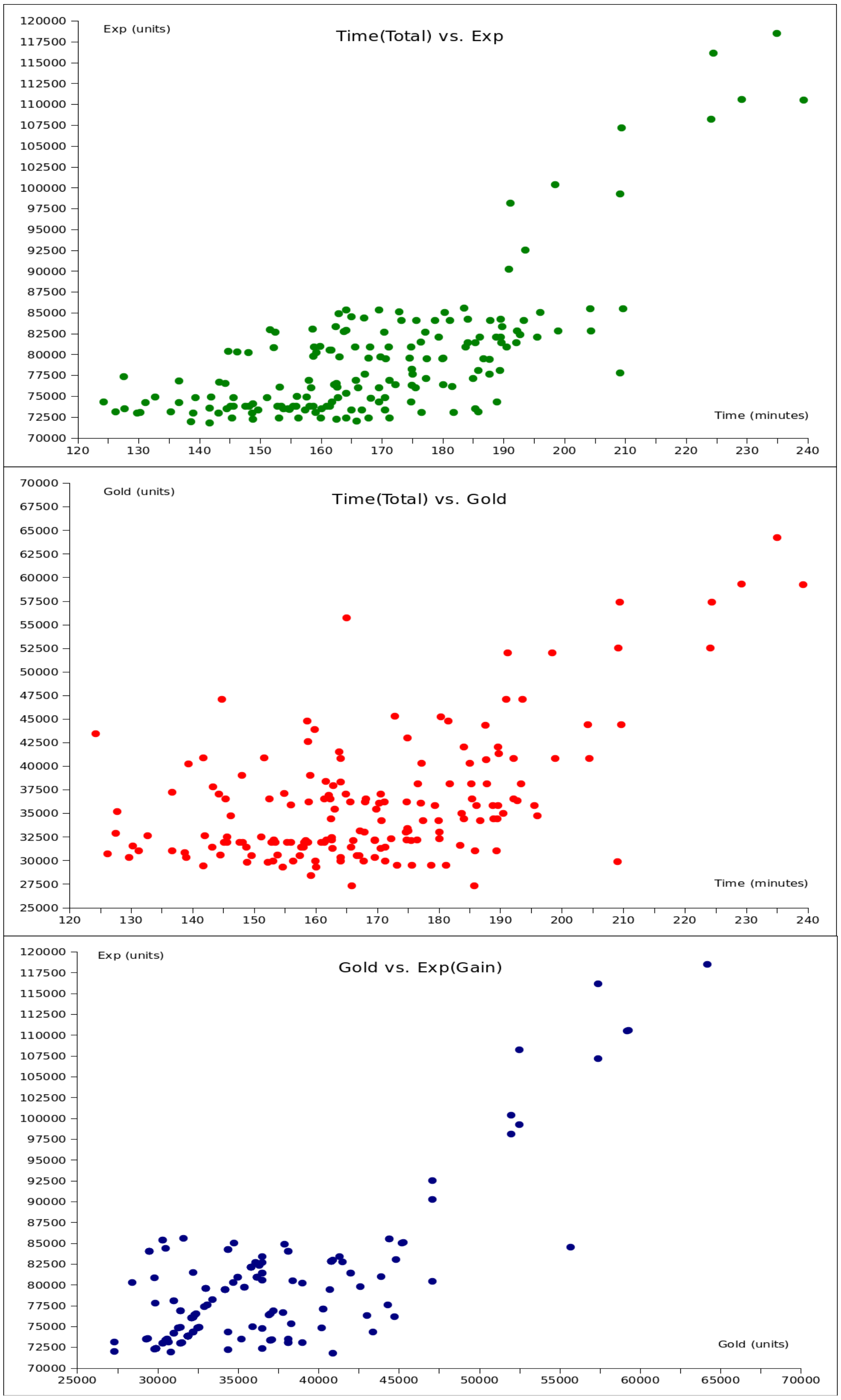

Time(Total) vs. Exp
Exp (units)
Time (minutes)
Time(Total) vs. Gold
Gold (units)
Time (minutes)
Gold vs. Exp(Gain)
Exp (units)
Gold (units)

Figure 6.3.1

6.4. A Few Discoveries

From the two experiments above, we also get something interesting about the game design rationale besides the pure balance problem.

Looking at the last line of the *_full.cvs* file in the program package, we happen to observe that nearly 50% to 60% of the quests from our quest system are not even touched by the player. In other words, over half of the game world is probably not explored at all!

The reason why this problem occurs, under our suspect, is that many other systems of the game affect the quests as a result of heavy interaction. A player might be able to level their characters up by do Grindings besides go Questing. This combination of the systems leads to part of the game being unexplored. And after repeated testing, this discovery is verified.

7. Related Works

During our investigation of the quest system balancing in MMORPG game, complexity is the main problem for us to spend most time fixing and improving. In many game researches these days, again, complexity often appears as a big obstacle, especially the case when running time is required yet the data set is huge with exponential growth.

A good example is the path-finding problem. Given a randomized, unexplored map with a start point and an end point, we want to find a path or a series of actions from start to end. On solving this, people usually begin with building a search tree rooted at the start. The search cost, depending on how people divide the map into smaller grids, can be quite expensive and unaffordable. Similar to our approach here, people sometimes use the selection algorithm to choose only “healthy” and representative paths across the whole search tree. This strategy reduces the size of the tree tremendously, and is very helpful when we are not necessary to find an optimal shortest path, but rather focusing on getting a result.

The Tic-tac-toe is another topic that involves both complexity and balance elements. It is a typical game that can be searched fully on its action tree, due to its tiny size of less than 9 branching factor each level. In that case, balance issue can be expressed completely. In fact, we do get proven result that the one who plays first is guaranteed to achieve at least a draw [2], which means the game is not balanced at all (unfair to the second player).

8. Future Works

Game balance is a topic that can be traced back in early time of game theory, and yet gets quite popular recently, especially games that focuses on data configuration so as to increase the accuracy and stability of their design. The Quest System analysis in our work provides a quick glimpse of PVE part in this area by investigating the problem with an analytical model. The next step, probably getting more challenging, would be an attempt from PVE related balance into the PVP oriented balance.

As for the state-based nature of our approach, a direct application could be the balance analysis of turn-based PVP games, such classic combinational games (See Reference [3]), or modern strategy games like **HEROES OF MIGHT AND MAGIC**, etc. By increasing the dimension and directions of our game search tree, these future works are possible, as long as they reside in an observable and real-time independent world.

9. References

[1] **"PERFECT WORLD INTERNATIONAL" (2008) WIKI.**
http://www.1perfectworld.com/wiki/International-Guides:Introduction

[2] Tic-tac-toe Strategy: http://en.wikipedia.org/wiki/Tic-tac-toe

[3] Richard J. Nowakowski, *Game of No Chance,* Cambridge University Press, 1998