Game Balancing

References:

Andrew Rollings and Ernest Adams on Game Design, Chapter 8.

http://www.xs4all.nl/~jbontes/ (Life32)

What is Game Balance?

- A balanced game is one where the main determining factor for the success of the player is the skill level of that player.
- Random events can occur, but a better player should be more successful than a poor one unless he has an unusually long run of bad luck.
- Game balance is very much a trial-and-error process:
  - → play game
  - → tweak game
  - → play game
  - → tweak game and so on
  - → time runs out, release game
  - → tweak some more in the form of patches
Balancing a Game

- Balancing a game is a very difficult concept to grasp.
- What are you balancing against?
- Are you balancing it against itself? The player?
- And how exactly are you balancing it?
- Are you balancing it so that it is a fair game?
- Are you balancing it so that it provides a consistent experience to the player no matter what her ability?
- The answers to these questions are somewhat subjective and depend on the nature of the game.

→ For example, a historically accurate simulation of the Anglo-Zulu wars would not be a fair game, since the Zulus would have to lose.
Balance

- There are two main kinds of balance:
  - static balance
  - dynamic balance
Static Balance

- **Static balance** is concerned with
  - the rules of the game and
  - how they interact with each other.
- These are time invariant.
- Example: the relative strengths of units in a real-time strategy game
Static Balance: Net Payoff Matrices

- Consider a game involving players red and blue.
- Each of these players has two strategies:
  - red has strategies R1 and R2
  - blue has strategies B1 and B2.
- Suppose the net payoff matrix is as follows:

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>R2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

- Negative numbers indicate a win for blue while positive numbers indicate a win for red. Zero indicates a tie.
- In a net payoff matrix, each value represents a net payoff, so, for example, the strategy combination R2 and B1 may not always result in a win for red but it usually does.
- In computer games, we deal with a large number of random events that do not always have the same result, but that will tend toward a certain value; hence, the net payoff tends toward a specific value.
Static Balance: Dominant Strategies

- A weakly dominant strategy is at least as good as any other strategy, no matter what the other player does.
- A strongly dominant strategy is better than any other strategy, no matter what the other player does.
Example of a Dominant Strategy

- Suppose you’re returning home from work, when suddenly you wonder whether it’s your wife’s birthday today.
- Should you buy flowers or not?
- Suppose the net payoff matrix (in terms of brownie points) is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Wife’s Birthday</th>
<th>Not Wife’s Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy Flowers</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Don’t Buy Flowers</td>
<td>-100</td>
<td>0</td>
</tr>
</tbody>
</table>

- The (strongly) dominant strategy is to always buy flowers.
- To turn this into a real game decision (by eliminating the dominant strategy):
  - we would have to attribute some cost to the flowers so that buying flowers when it is not her birthday results in a negative payoff
  - we’d also have to figure out the exchange rate between dollars and brownie points
Static Balance: Dominant Strategies

- Here’s an example from the real-time strategy game Red Alert.

- The game is well known for a particular dominant (or near dominant) strategy, which has become known as the tank rush (a name that has spread to similar strategies in other games).

- An experienced player playing as the Soviet side could devote all of her energies to producing a large force of tanks in the early part of the game, and then use those tanks to attack the nascent enemy base en masse.

- Against an unprepared opponent, this almost always guarantees a victory.

- Because of the immense number of variables involved, we cannot say with certainty that this is a dominant strategy but is certainly near dominant.
Static Balance: Symmetry

- **Symmetry** is the simplest way of balancing a game.
  - Each player (including the computer) is given the same starting conditions and abilities.
  - The outcome of the game now depends only on the relative skill level of the players (though of course, in a turn-based game like chess, one player may have an advantage depending on who makes the first move).

- While this approach works in abstract games such as chess, can you imagine simulating a real battle with the infantry, knights, and archers lined up facing each other over a perfectly symmetrical field?

- It would feel unnatural and would quickly become boring.

- Note however that pure symmetry works very well for sport simulations.
Static Balance: Symmetry

- Instead of pure symmetry, we can have functional symmetry.
  - Functional symmetry is a form of symmetry where the abilities of the player are roughly — but not exactly — mirrored.
  - That is, they are functionally equivalent, not exactly equivalent.

- Consider a real-time strategy game:
  - first player starts behind a mountain range
  - second player starts behind a river
  - first player has the ability to create boats
  - second player has the ability to create helicopters
  - the helicopters can pass the mountain range
  - the boats can pass the river
Static Balance: Symmetry

- symmetry is mainly useful in balancing multi-player games
- but it can also be used in balancing single player games:
  - it’s a good starting point to ensure that the computer-controlled entities are roughly in balance with the player
  - however, the player runs the risk of being restricted to an simple game of tit-for-tat: the computer throws X at me, so I have to respond with Y, which encourages the computer to throw more X at me, and so on; this does not lead to particularly interesting gameplay
Static Balance: Transitive Relationships

- In a transitive relationship, if A can beat B and B can beat C, then A can beat C.
- Why would anyone want to use C when they could use A?
- Give them a reason by balancing:
  - → A comes at a higher cost (e.g., not necessarily direct dollar amounts, but indirect costs in terms of lives of people sent to fetch the entity, in terms of the difficulties that the player has to endure, etc.)
- These indirect costs are called shadow costs; they should be sufficiently high as to justify the reward.
- Shadow costs are the end result of a number of other factors: they can be measured, but cannot be directly modified.
- Pure transitive relationships, without shadow costs to balance them out, lead to trivial and undemanding gameplay choices, which serve to undermine balance.
Static Balance: Transitive Relationships

- Transitive relationships are very common in games, especially games such as first-person shooters.

- For example, in Doom:
  
  → player starts with a relatively weak pistol
  
  → a little way into the game, the player finds a shotgun, which is much more effective
  
  → now the monsters become harder to kill and the shotgun ammunition becomes scarce
  
  → we need to search for another stronger weapon to make the same progress we were making before
  
  → and so on...

- Any game that involves upgrading or augmenting the player’s capabilities makes use of transitive relationships.
Static balance: Intransitive Relationships

- Consider the children’s game Rock, Paper, Scissors (sometimes called Scissors, Paper, Stone):
  - $\rightarrow$ scissors cut paper (so scissors beat paper)
  - $\rightarrow$ paper wraps rock (so paper beats rock)
  - $\rightarrow$ rock blunts scissors (so rock beats scissors)

- This gives us a balanced, 3-way intransitive relationship.

- The 3-way intransitive relationship of Rock, Paper, Scissors has been the model for most real-time strategy game balancing and has also been used in other game genres such as racing games and role-playing games.
Static balance: Intransitive Relationships

- Problems with static intransitive relationships:
  - It’s too easy for the player to learn the simple relationships between units and figure out the best strategy to use.
  - They can lead to uninteresting gameplay because each strategy tends to be used equally, and the pattern can become predictable.

- In order to make the decisions more interesting, we can vary the shadow costs to alter the likelihood that the particular strategy/unit is chosen.

- Example: We have a confrontation between crafts A and B:
  - Both A and B can operate as an aircraft or submersible.
  - A is optimized for flight and tends to beat B in air combat.
  - B is optimized for submersible operation and tends to beat A in submerged combat.
  - By altering the environment (and hence, the shadow costs of operating the craft), we introduce an interesting dynamic in the relationship between crafts A and B.
Static Balance: Trade-Offs

- Sometimes one entity might be better than another in some ways but not in others.
- This is common in role-playing games:
  - The player can distribute a limited number of points among a number of attributes, such as strength, stamina, and intelligence.
- This point distribution could also be done behind the scenes.
  - For example, consider a simple platform game where the player is pitched against a number of different enemies.
  - For each level, the enemies could have a fixed number of points to divide between two attributes, speed, and jumping power.
  - This would give us a nice range of enemies of different abilities.
Static Balance: Combination

- In some cases, two or more entities can be treated as a single entity when balancing a game.
  
  → For example, even though one unit of infantry may not be enough to beat one archer, you might be able to use a unit of infantry in combination with another unit.

- As a designer, you wouldn’t necessarily be expected to explicitly design in all the possible combinations of entities within your game.

- You do, however, have to be aware of them and to balance the more troublesome ones by modifying the entities themselves and using shadow costs to equalize them.
  
  → For example, the previously mentioned tank rush from Red Alert could have been avoided by modifying the shadow costs of tanks so that they were much more expensive to produce in the early stages of the game.
Static Balance: Emergence

- **Emergence** is the action of simple rules combining to produce complex results.
- Classic example: Conway’s Game of Life
- We start out with a configuration of living cells (organisms) which are placed on a 2D grid.
- This constitutes the first generation.
- Rules are then used to get subsequent generations:
  - → Any live cell with fewer than two neighbors dies of loneliness.
  - → Any live cell with more than three neighbors dies of crowding.
  - → Any dead cell with exactly three neighbors comes to life.
  - → Any live cell with two or three neighbors lives, unchanged, to the next generation.
- Even though the rules are simple, some very interesting patterns emerge.
- Try it and see.
Static Balance: Emergence Example

- Imagine that our character is on one side of a locked wooden door and wants to get to the other side of the door.

- Some solutions:
  - find the key and open the door
  - pick the lock
  - try and cast our magic Open Sesame spell and open the door magically

- More obscure solutions (demonstrating emergence):
  - break down the door with an axe
  - burn though the door using fire or acid
  - cast a spell to turn it to stone or glass and then shatter it
  - unscrew the hinges or the lock
  - use acid to burn though the hinges or the lock
  - break our way through the wall next to the door
  - cast a “ghost” spell on ourselves that lets us pass through solid objects
Static Balance: Emergence Example

- To get these more obscure solutions to work, we can design our simulation so that we take into account a limited subset of fundamental properties of matter and handle our interactions between objects in that fashion.

- Then we could consider the door as a collection of connected objects:
  - the hinge
  - the door itself, and
  - the lock

- the door is made of wood; wood has a set of properties:
  - flammability (high)
  - resistance to acid (average), and
  - strength (average)

- the lock and hinges would be made of metal; metal would have a set of properties:
  - flammability (low)
  - resistance to acid (poor) and
  - strength (high)

- similarly, you could assign properties to the stone wall holding the door that detail whether it could be broken through or not
Static Balance: Emergence

- Emergence has to be used with care.
- The player could imagine ways that we haven’t considered in order to get through the door.
- This makes it difficult for us to control the gameplay directly.
- Emergence has its dark side:
  - it can undermine gameplay, leading to an undesirable dominant strategy or, worse, fatal gameplay flaws
Static Balance: Feedback Loops

- The basic progression of a game is that it starts statically and dynamically balanced and then gets out of balance, first one way and then the other.

- It goes backward and forward like a seesaw with one player ahead and then the other, until someone eventually gets so far ahead that it is impossible for the other to catch up.

- There are two types of feedback loops:
  
  \[ \rightarrow \text{positive feedback}: \text{makes it easier when you are ahead} \]

  \[ \rightarrow \text{negative feedback}: \text{can be used to counter positive feedback} \]

- Another solution for too much positive feedback is to include a random factor that gives the player who’s behind a chance to catch up just through sheer luck.
Positive Feedback

- Often, being ahead makes things easier for the player in that position and harder for the other player. This is positive feedback that helps the leading player.
  \[\rightarrow\] e.g., in Monopoly, the more money you have the more hotels you can put up, which produces more money, and so on.

- This is a desirable trait as long as it doesn’t happen too fast and doesn’t leave the player who’s behind with no way to catch up.

- You want positive feedback so that the game will end eventually, but not too much
  \[\rightarrow\] Example: Warcraft does not let you take and use enemy factories; you can only damage and destroy them. If you could use enemy factories to build armies, the game would become unbalanced too quickly.

- A game where the slightest advantage leads to a runaway victory for one player would not be fun.

- If you are going to use positive feedback loops in your design, make sure they have a reasonable response time delay before they kick into action.
Negative Feedback

- You can counter positive feedback with negative feedback (in addition to the response time delay).
- Suppose taking an enemy piece enables you to use it, but there is a price to be paid for it: It must be supported somehow.
- This means that taking it is not “free”.
- For example, in Dungeon Keeper:
  → You could torture enemy creatures to convert them to your side, but once you did, they had to have food and money and a place to sleep.
  → The process of converting them also took time, and if you weren’t careful, you might kill them without converting them.
Dynamic Balance

- with dynamic balance, we are concerned with how the balance changes with time and player interaction.
- a game system should initially be in a state of static balance, but once it is set in motion, a different form of balance, the dynamic balance is maintained.
Dynamic Balance: Challenge

- The game should scale in difficulty smoothly as the player progresses into it.
- In some games, the midgame experience turns out to be substantially more difficult than the endgame! This is a bad idea!
- The player would feel that anything after that difficult point would be anticlimactic.
Dynamic Balance: Fairness

- A major factor in whether a player enjoys a game is whether she **perceives** it to be fair or not.
  - It does not actually matter whether the game is fair.
  - What is important here is the player’s perception of fairness.

- You could allow computer opponents to cheat (perhaps because it will make the implementation easier, take less resources, etc), but you should do so subtly.

- Blatant cheating by the computer is seen as a sign of laziness on the part of the designers and developers.
Dynamic Balance: Fairness

- No good designer would knowingly design a game where a player destroys all changes of winning by taking an action earlier in the game and not finding out until later in the game.

- Example:

  → In Monty on the Run, the object of the game was to guide the hero to freedom and to escape the long arm of the law.

  → When the player started the game, he had to choose five items to take along.

  → These items would help get past various obstacles throughout the game.

  → The problem was that the player was given no clues as to which were correct and which were not.

  → Thus, the player was effectively doomed from the start unless the correct choice was made at the beginning of the game.

  → This is clearly not fair.
Dynamic Balance: Stagnation

- **Stagnation** occurs when players become stuck.
  
  → Example: running around a level of a first-person shooter trying to find the last hidden switch that opens the level exit.

- Two ways to tackle stagnation:

  - **Passive.** the designer can make sure that the clues about how to proceed are hidden in plain sight

  - **Active.** have the game work out whether the player has been wandering around aimlessly and provide a few gentle nudges to guide him in the right direction

- If many players need to resort to outside assistance (e.g., looking up answers on the web), then this is an indication of a failed game design.
Dynamic Balancing: Trivialities

- Don’t give the player *trivial decisions*!
  
  → Example: Don’t force the player to decide where the gold is stored when she is trying to build an army and plan a grand strategy!

- A *trivial decision* is one where there is one logical outcome, or where the outcome has no real effect on the game.

- The player should not be bothered with these.

- Let the computer handle it and, if necessary, inform the player afterward.
Dynamic Balance: Difficulty Level

- Players can often choose from three or four difficulty levels. For example:
  - easy
  - normal
  - hard
  - nightmare
- To increase difficulty, you can make enemies:
  - tougher (e.g., they require more shots to kill, they have more deadly weapons, etc.)
  - numerous
  - smarter
- Some games (e.g., Max Payne) dynamically adjust their difficulty level (i.e., while the game is in progress) to something appropriate given the observed skill of the player.
- But dynamic difficulty level adjustment can lead to abuse of the system.
- For example, a skilled player could deliberately play badly just before he gets to a really tough section of the game so that the game will go easier on him when he gets there.
Balancing Technique: Design for Modification

- Design a core set of rules that the game adheres to and then design the game entities to conform to those rules.

- As long as the core rules are balanced, tweaking them slightly will probably not affect the balance in wild and unpredictable ways.

- Example:
  - In Age of Empires, all the game entities are governed by the same rules.
  - They have a large set of parameters used to configure those rules to distinguish each entity class.
  - A change to one of these parameters does not require a corresponding code change.
  - This allowed the designers to tweak parameters easily.
Balancing Technique:
Tweaking and Experimental Methods

- Tweaking parameters randomly is an inefficient and wasteful way to modify balance.

- Some tips:
  \[\rightarrow\] modify only one parameter at a time and see what happens
  \[\rightarrow\] when initially modifying parameters, don’t bother with small changes (e.g., initially, try doubling or halving the parameter and see what happens)
Summary

- A balanced game should:
  - be internally consistent
  - ensure the victory is determined by player skill, not random factors
  - ensure that all players have access to the same or functionally equivalent core options
  - ensure that combination and emergence don’t destroy the balance
  - provide a consistent challenge
  - provide the player with a perceivably fair playing experience
  - avoid stagnation
  - avoid trivialities
  - allow setting of difficulty level (where appropriate)