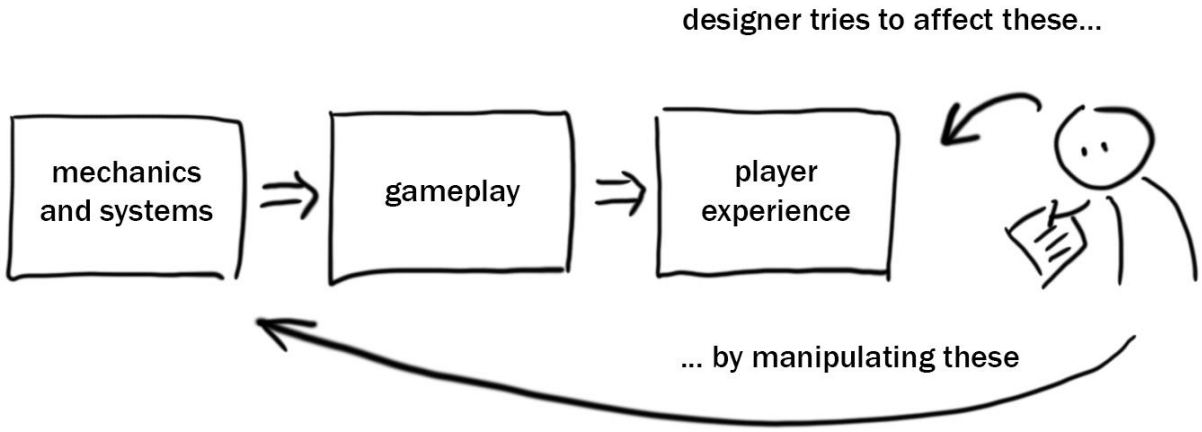
Mechanics

After discussing player experience in (TODO REF CH2), we can now start looking into how to bring about these kinds of player experiences.

The designer's role is restricted by the medium. Recall our initial diagram from (TODO REF CH1):



We will now wrap around, and restart our discussion from mechanics, and then move on to discussing how they produce gameplay.

This chapter will talk about mechanics separately from other design aspects, and even separately from each other. In the *next* chapter (TODO REF CH4) we will discuss how they all work together, and form systems for the player to interact with. But for now, we will introduce them by themselves.

So, what are these "mechanics" that we keep bringing up?

Mechanics as building blocks

The term "mechanics" describes the collection of basic elements that define how a game is played on a fundamental level. There is no exact definition of the word, and since the term has been in use for a long period of time in various game design communities, it is not uncommon for well-intentioned people to disagree about what exactly constitutes a mechanic.

For our purposes, we will keep the idea simple. Mechanics are the most basic elements, and these typically come in three broad types:

- The *pieces* that can be acted on or manipulated.
- The kinds of *actions* that can be performed with pieces
- The *rules* that determine what actions can be done when, with what pieces, and what the outcomes will be.

We can describe them in greater detail:

1. Game pieces are likely the most intuitive element. Physical games are full of concrete pieces to manipulate. Poker has chips and cards, the table and the pile of money in the middle of it. Chess has the game board and board pieces, and sometimes also a game clock. *Settlers of Catan* has the game board, improvement tiles, resource cards, and dice.

Games can also have *virtual pieces* that represent physical ones. The character sheet in a role-playing game is full of virtual swords and battle axes, gold pieces, potions and spell books, which are only symbols on paper (or screen), but they function as pieces that can be manipulated.

Game pieces can also be *abstract*. Player's "hand" in poker is just a grouping of other pieces, but it's sometimes useful to treat it as a single "thing". Also, poker and chess and *Catan* all have a notion of a "turn" that is in possession of one of the players, which is meaningful because most actions are only available to the player who has the turn, and yet there is no object, physical or virtual, that corresponds to it.

2. Game actions are the flip side: if we have game objects, we should be able to do something with them. Poker players draw cards, ante up, call others' hands or perhaps fold their own. Chess players move their pieces on the board, take the opponent's pieces or lose their own, and sometimes exchange pieces like with pawn promotion. *Catan* players roll dice to randomly choose a tile to harvest, collect resources, trade, and build improvements.

There is an infinite number of actions that could be defined by the game designer. Actions can affect pieces (move them on the board, modify their attributes), exchange them, remove them or produce them, and so on, in a myriad ways that are specific to the game.

3. Game rules then specify how exactly those actions are allowed to work. Chess rules for instance, specify how the specific pieces are allowed to move on the board, or how piece capture is allowed to work, in standard circumstances as well as *en passant,* and so on.

Rules, actions, and pieces are all intertwined. When we talk about "drawing a card" in poker, that describes an action and relevant game pieces. But understanding this also relies on knowing the rules that define and constrain the situation: that only the player holding the turn can draw a card, just a single card, always from the top of the deck, the card goes into the player's hand, and so on.

Additionally, rules also often describe other things about the game that might not be mechanics. For example, rules of chess define the winning and losing conditions for a single game of chess, while ranked match rules define how to compute one's ranking score. Victory conditions are not mechanics themselves, not in our understanding of the word: they are the goals towards which game players try to achieve using mechanics, and they form a part of a *metagame* which we discuss in (TODO REF CH6).

In short, we can think of game rules as a large set of various regulations about the game. They including rules for how actions and pieces interact, but also any other additional regulations that are desired.

Mechanics as nouns and verbs

Another common metaphor for game mechanics is to compare them to human languages, and specifically to language syntax:

- Game pieces are like *nouns,* since they describe things
- Game actions are like *verbs,* since they describe what happens
- Game rules are like *grammar,* since they describe how nouns and verbs can be put together

This is a very useful metaphor, as the similarities are very close. Beyond just the structural similarity, there is also a learning similarity: humans often learn how to play a game from just observing and participating, without having to have all the rules spelled out for them, similarly to picking up a language (but this is common with rule-oriented activities, we are all quite adept at inferring rules).

This metaphor also gives us ideas that there may be other types of mechanics to consider:

- *Adjectives* that modify nouns: a *weak* armor compared to a *strong* armor
- *Adverbs* that modify verbs: jumping *high* compared to jumping *low*
- *Prepositions* that modify the relationship: sitting *at* the table compared to *under* the table

We will see the equivalents of these modifiers later on (TODO REF SECTION) as we get into the details of mechanics.

Example: exploring Monopoly

To illustrate this point we can look at the game of Monopoly, and see what kinds of mechanics we can identify. We have already discussed the main categories:

- *Nouns that a player can manipulate:* a game board, dice, pawns, cash, properties, upgrades, sets of random event cards, etc.
- *Actions a player can perform:* rolling dice, moving counterclockwise, landing, buying, paying rent, drawing random cards, etc.
- *Rules that govern the game:* which actions can be performed given what context, what resources are needed, spent, gained, and what happens as the effect.

We can try to tease out the individual nouns and verbs by iteratively questioning the game about how it works:

- Who am I as a player? I'm a game piece on the game board.
- What do I do? I move on the board, and then react depending on where I land.
- How do I move? I roll two six-sided dice and move clockwise that number of tiles.
- What do I do after landing? It depends on the tile type.
- If it's someone else's property? I pay rent based on property characteristics.
- How do I compute this? There is a base rent that changes depending on whether it's part of a set, and whether it has improvements.
- What is a set? A collection of adjacent tiles of the same color, owned by the same person.
- What is an improvement? A house or a hotel placed by the owner.
- How does this affect rent? A chart determines how rent changes due to sets or improvements.
- … And so on. There are many, *many* more details we could list out.

As we can see, this kind of a Socratic dialog with a game artifact can get pretty detailed. Also, when we are making a new game, it is absolutely necessary to figure out all the pieces, actions, and rules at such a high level of detail, and commit them either to a rule book, or encode them in computer instructions. We will come back to this topic in the next chapter (TODO REF CH4) after discussing systems.

However, in the discussion coming up next, we will not look at the *individual* objects and actions, but instead we will look at entire *types of actions* and *types of objects*. For example, Monopoly uses two six-sided dice to move around the board: this is a very common pattern in games, both the part about using dice as a source of randomness, and using randomness to traverse a game board. This is the level of abstraction we want to focus on when we describe mechanics.

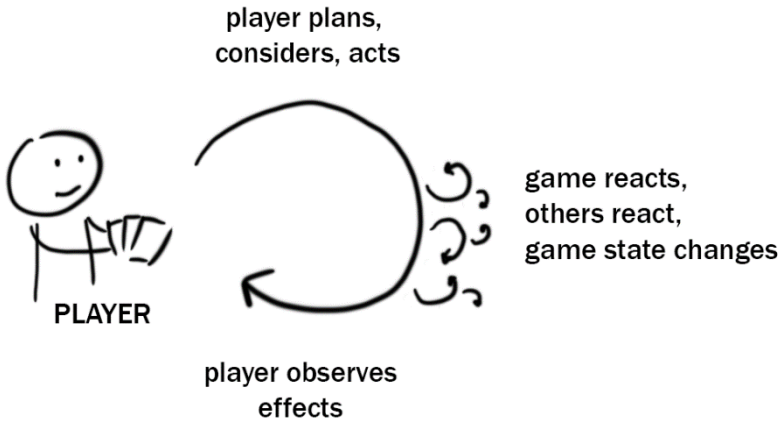
Core terminology

We will turn to some concrete examples of mechanics in the next section. But first, let's quickly review some of the common terms that game designers use when talking about game mechanics, such as game state and action space.

Game state

As Sid Meier said, a game is a "series of interesting choices". (TODO REF) The crucial element of game design is *player's choice and how it changes the game over time.* A game of poker or chess is not static – as players perform actions with the game pieces, the situation and players' fortunes change quickly.

To describe the influence of time, designers often borrow terminology from cybernetics and automata theory. Players and the game are the parts of a dynamic system, and the player's actions change the state of the system, which in turn causes the players to adjust their decisions and take more actions, which further change the state, and so on.



In other words we can say that:

- *Game state* is the total state of all game pieces and their properties at a point in time
- Game actions are *transitions* between game states

Game state may be fully or partially observable. Chess is an example of a game with *fully observable* state: if we look at the state of a chess board at some point in time, the game board fully describes the current state of the game. We can take a picture of how the board looks, then reconstruct it at another time or in another place, and continue the game as if it had never been interrupted.

Most other games have hidden information, however, which means the current state is only *partially observable*. In a game like Monopoly, some information is available to everybody (e.g. which tiles are owned by which player), some is only available to a single player (which random event card I have), and some is hidden from all players (the order of random event cards in the pile). But all this information comes together to make up total game state. If we wanted to save the game state to reconstruct it later, the hidden information would need to be saved as well.

State spaces

From this perspective, a game is a series of game state transitions over time. In chess, the number of possible transitions is huge – the player often has many possible actions to choose from, and the game *could* transition into various possible states. But in the end the player must choose just one, and live with the consequences.

Looking at all the *possible* states the game *could be* in, we can add a few more definitions:

- The *state space* is the set of all states that the game could be in, connected by actions that transition from one state to the next
- The *state space size* is the total number of states in the state space, or in other words, the number of different states that could be reached from the initial game state
- The *branching factor* for a state is the number of transitions (actions) out of that state

Informally, we can talk about state spaces being *dense* or *bushy* when each state has a relatively high overall branching factor, or *sparse* when it's relatively low.[1] What counts as "dense" is relative to the genre and expectations: for example, children's board games are not expected to be nearly as dense as strategic board games for adults.

Space density and action set size both affect how players perceive the game. A sparse game that does not offer a lot of interesting decision points (say, Tic Tac Toe) will not be as interesting as a denser game. However, this is again very audience-specific: younger children love Tic Tac Toe because their cognitive skills are still developing, so the complexity of Tic Tac Toe is a better match. Similarly, in the other extreme, a dense game might be more interesting, but *too dense* of a space will overwhelm a new player with choices they don't yet understand. Frustratingly enough, the question of "how dense is too dense" is very subjective.

[1] Some designers (Elias et al REF) also speak of "game trees", which are game state spaces that contain no loops – in other words, when the game state can never repeat itself, such as in Tic Tac Toe, or in many card games. This is because in those games, the game state forms a directed, acyclic graph, which are traditionally called "trees" in decision theory as well as computer science. Admittedly, the term "bushy" makes much more sense when applied to trees than to graphs.

Even in sparse games, however, the state space sizes are routinely incomprehensively large. Size calculation for the simple game of Tic Tac Toe seems straightforward: 9 choices followed by 8 choices followed by 7 choices, etc. But this means the space size has a factorial upper bound of *n = 9! = 362880* (although that includes invalid states, so the actual space size will be smaller).

At least the state space of Tic Tac Toe can be exhaustively enumerated and searched. The state space of chess or go has such a huge branching factor, that it is impossible to search exhaustively even by the most powerful computers (and chess and go playing software must resort to shortcuts and heuristics to try to cut down the state space it has to reason about).

Action spaces

State space contributes to the strategic complexity of a game, but how these states are connected *via actions,* or the *topology* of the state space, influences the player's feeling of agency and what they can do in the game world.

Action space defined formally is the set of all actions that are available to the player. However, game designer tends to abuse the term informally, to mean the set of *actions available at the present state*.

For example, in chess, the overall action space would be the set of all possible moves ever, but informally, the current action space is just the set of moves available to the player right now. If players sit down for a new match, white gets the first move, but their action space is tiny: eight different pawn moves, and just four possible knight moves, out of a much larger set of all possible actions.

In this text, we will use the informal term *action space* to refer to the contextual set of actions available in a given state, and use the term *overall action space* to mean all actions.[2]

Action space size is very important. In a game like Tic Tac Toe, the initial action space consists of nine possible moves, then reduces to eight, then seven, and so on – which is ever-decreasing and not very satisfying, compared to a game like chess (or even checkers) where the action space is much more complex.

Secondly, as we already discussed, action space can vary over time, which is very interesting for the player. In chess, the action space in the beginning is very limited, but it "opens up" in mid game, as the player expands their control over the game board, only to collapse back down in late game.

Part of the enduring appeal of many games is the challenge of *controlling future action space*. In other words, planning well ahead to make sure we do not "box ourselves in", where our situation is suboptimal because of poor choices we made early on.

[2] Some designers try to avoid this ambiguity, for example Upton (TODO REF) uses the term *action horizon* to describe actions available immediately and in the near future, while others (Wright REF Rules of Play REF TODO) use the term *possibility space* to describe the combination of actions and states contextually available to the player.

Perceived action spaces

Action space describes actions that the game makes available to the player. However, player's *perception* of the action space is just as important. For some examples:

- As a player, I might not realize some actions are available: as a new player in a combat game, maybe I have not yet learned how to perform blocking maneuvers, so I do not even know that I should consider them.
- I might know about actions but not consider them, because they do not fit my goals or strategies: I could shoot myself in the foot instead of shooting my enemy, but why would I even consider that?
- I might also think some actions are possible even when they are not: I might fear that a false step will send me falling off of a cliff, when in reality there is an invisible barrier there.

We can describe this as the player's *perceived action space,* that is, the set of actions they think they can or should perform at the given time.[3] The player's real and perceived actions can diverge, but they must overlap at least somewhat, otherwise the game will be unplayable. If the player fails to perceive what they can do, they will not be able to play the game, and if they consistently perceive actions that are not available, they will grow frustrated. Tutorials typically try to teach the player the basics about how to perceive available actions, but not all of them, just enough to bootstrap their own learning.

It is important that player perceive a large action space. At the same time, just having a large number of possible actions is not enough: those actions should also be interestingly different from each other, and the more significant the differences, the better. For example, we could imagine the following spell types in a role-playing game:

- Spells that have different names, but the same cost and benefit, or
- Spells that have similar but different costs and benefits
- Spells that have vastly different cost and benefits

The latter ones will be increasingly more interesting to the player. Choices in the first case are *illusory*: there may be multiple surface manifestations of the options, but they all lead to the same outcome, so they all represent the same transition in the state space. In the latter two cases, the choices are very different, which will push the game in very different directions – and the larger the differences, the more interesting the choices become. It is crucial to keep the player's action space well differentiated, with non-obvious consequences.

We will return to talking about action space design heuristics later on in (TODO REF CH5).

Explicit and Implicit Mechanics

While it is possible to write out the rules for *Monopoly* or chess, it is much harder or even impossible to write out the rules governing a computer game, whether a strategic game like *Civilization* or something more action-oriented like *StarCraft*, or especially an arcade game like *Tekken*.

[3] Similarly to the previous footnote, Upton (TODO REF) uses the term *intent horizon* instead.

This is the distinction between *explicitly* and *implicitly defined* mechanics. Explicitly defined mechanics are what we would encounter in a physical game, like *Monopoly* or poker – all actions and pieces are described in the rule book, and the player can see everything that the game has to offer on the box or on the table in front of them. In these games, players play the game knowing all of the mechanics (and even if they do not remember them, at least they are fully accessible and can be looked up).

Implicitly defined mechanics are much more common in video games. These rules are not known to the player ahead of time, and often *only* exist in the computer code that implements them so they cannot be inspected. When the player starts out, they only know some mechanics (from the tutorial, or watching other players) and as they play the game, they discover the rest, and learn how they all tie together. For example, in a combat game, the player may discover how exactly attacks need to be timed by practicing and learning from mistakes, or in a war game, they may run into a new enemy with unit types that they have never seen before, and now they have to figure out how to counter them.

Implicit mechanics can offer a high level of surprise and discovery that is difficult to match in fully explicit games, and the learning element can be very rewarding. In those kinds of games, the player is learning what kinds of mechanics become available over time and how to best use them to their advantage. However, this effect is sensitive to implementation details, as this can make the game far more challenging than one with explicit mechanics which are plainly visible to everyone.

Some games exist in between those two extremes. For example, physical collectible card games like *Magic the Gathering* have fully explicit rules for the game, but each card can also extend them – and since there are scores of rare cards, and players need to buy those cards separately to access those rules, we can treat them as mostly implicit. Similar situation happens with board "legacy style" games that come with secret cards or rule changes that must be opened at a specific time in the game. On the other hand, for a contrasting example, a computer war game might be usually played as if its mechanics were implicit, having the player discover them as they play through the campaign, but an extensive manual might also exists for hardcore players who want to take the time to research them in depth.

The difference between explicit and implicit rules is also the reason why it is difficult to use paper prototyping for computer games. Paper games work for prototyping explicit mechanics, but they are a very poor fit for prototyping implicit mechanics, and especially if those mechanics are based around physical activity or complex resources.

Examples of families of mechanics

Armed with some shared terminology, we can now discuss a number of examples of game mechanics, across different kinds of games.

An unknown number of mechanics have been created, to say nothing about mechanics that are yet to be invented, so we cannot possibly try to enumerate all of them. As an industry, we even lack a shared, common taxonomy of different types, the field of game design is too young to have settled on one, although some work has happened in this direction (see Further Reading).

Even so, we can talk about mechanics in terms of broad categories. In this section we will discuss four families of mechanics in greater detail, and then mention some more families that will be discussed only very briefly.

The following families of mechanics are very common, and appear in a variety of games across time and genres, so we will discuss them in a big greater detail:

1. **Control mechanics:** how the player is represented in a computer game, and how the player's physical inputs are translated into controlling the game, their character, or other aspects.
2. **Progression mechanics:** giving the player feedback about their own performance in the game, and changing the game in response to the player's progress.
3. **Uncertainty mechanics:** the role of uncertainty (not knowing what will happen next), the variety of the different sources of uncertainty such as randomness or hidden information, and how that affects the gameplay.
4. **Resource management mechanics:** resource manipulation and ownership, such as mechanics around items, currencies, or resources that can be owned in the game, how things can be converted from one to another, or perhaps used up and consumed.

There are of course many more types of mechanics – such as those specific to play formats, genres, or platforms. Here is a quick list of some examples:

- **Board game mechanics,** such as those around space control and ownership, or rules around moving one's pieces on the game board
- **Wargame combat mechanics,** such as those found in wargame board games, which describe how the different kinds of military game pieces act on each other, role of terrain or cover, and even more abstract military concepts such as the role of surprise on determining outcome.
- **FPS mechanics,** which are specific to first-person combat games, and deal with weapons and their properties, detailed input management such as how aiming and navigation works, or the different types of team roles, goals, and dynamics.
- **RTS mechanics,** for games where players manage large numbers of moving units as well as production chains that manufacture them, and the mechanics describe the details of how units can be directed on a micro level and what they do, the internal economy of base buildings and how they convert resources into new units, or how terrain affects unit actions.
- **Racing mechanics,** such as those that represent how a car racing game might behave, including a detailed physical simulation of the vehicle, and how it responds to the player's actions, to the race track and weather conditions, and so on.
- **Multiplayer mechanics,** which concern issues around balancing the game to be fair to all players, yet introducing different challenges, such as by introducing asymmetry which then quickly leads the discussion towards mathematical game theory and the Nash equilibrium.
- … and many more

These latter mechanics families overlap quite a bit with each other, and it would be interesting to pursue them in depth. However, we will not discuss these latter kinds of mechanics here, since they are very numerous and context-specific.

Instead, let us look at the four examples mentioned at the beginning of this section.

Control mechanics

Many computer games are oriented around avatar action: the player controls an avatar that fights, runs, flies, shoots, and acts inside the game world. This is a shared element of games across many genres, from fighting games like *Tekken* or *Soul Calibur,* to platformer games like *Super Mario* or *Prince of Persia,* arcade games like *Pac-Man* or even *Asteroids,* and many others.

Action-oriented games tend to reward quick reflexes and player's accurate, masterful performance. Consequently, if the game's handling of player's actions is clunky and inhibits their performance, that will be very perceptible to the player, and it will feel wrong. Even if they cannot explain quite what is wrong with the game, players may complain that the controls feel "floaty" or "mushy" or "jittery", that the camera always gets in the way, or they may be getting frustrated and not even know why.

To make the game "feel" good, make it feel responsive and easy to control, designers pay special attention to the "three Cs" of action games: character, camera, and control:

1. **Character:** who or what is the player's avatar, what it can do in the game world, and whether the actions make sense, are fun to engage with, and present a challenge.
2. **Camera:** what the player can see and how well the camera follows the action, showing everything from the most useful angle and not getting in the way.
3. **Control:** how well the player's game controller input translates into character actions.

The rules and attributes that define these behaviors are commonly called *control mechanics* or *movement mechanics* or *combat mechanics*, depending on the particular focus.

The first two Cs are probably the most intuitive. Some games have a *character avatar* that represents the player, and if the character cannot do interesting things in the game, or if the camera's movement get in the way of the action, it is easy to imagine the negative experience this would generate: frustration, boredom, or a feeling of not being in control of how the avatar behaves, which works against masterful performance.

Cameras are another element, and they can be first person (looking through the avatar's eyes) or third person (looking at the game world from the outside). First-person camera can only see ahead of the character, and the limited field of view can be frustrating. A third-person camera can see more, but issues around camera movement are especially tricky: typically, we want to make the camera move by itself and follow action in the game, so that the player is not burdened by having to manage the camera while also playing the game. But making the camera "smart" when needed, making it look at the right thing at the right time, yet also be controllable by the player, is quite difficult and a huge amount of work goes into making camera movement so good that it becomes unnoticeable.

Control is the third element: how the player controls their avatar. This means figuring out what the player's inputs are going to be, and how they will translate into action in the game world, in a way that is seamless and puts the player fully in control. For example: in a boxing game, does the player hit a button to throw a punch, or maybe use the joystick to simulate moving their fist back and forward? Or maybe some other option, like having to move the avatar while also pressing buttons to pick between a high punch and a low punch? Which will feel better to the player, and why?

Game feel

In any action game, we need a way to translate from controller inputs into in-game actions, and typically there will be many more actions available than inputs on the controller. In which case, we will be faced with some tough choices, how to map from the player's physical actions to in-game actions. For example, in a fighting game, we may need to figure out:

- Which buttons/inputs are best for which actions and why? What is it about their positioning and sensitivity that makes them better or worse fit for specific actions? (For example: when to use thumb buttons, vs shoulder buttons, vs pressing on the analog joysticks, and so on)
- Do we use digital buttons that can only be on or off, or analog joysticks that give us a continuous measurement? Which game actions benefit from analog vs digital?
- Should we use chords? (eg. A+down for a low kick, A+up for high kick)
- Should we use combos? (eg. A-B-A in sequence for a triple kick)
- Should we use timed actions? (eg. press and hold A to "wind up" a forceful kick)

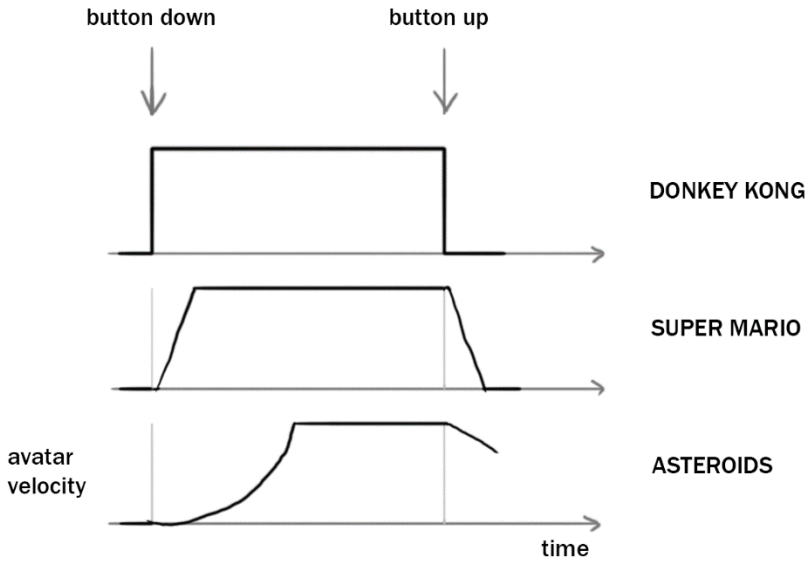
This kind of input mapping happens over several steps, and the game developer must make appropriate decisions at each step. Players will also come to a game with prior expectations, from other games in the genre or from how a game console standardized some input mappings, so those prior expectations need to be taken into account as well.

Once we get past input mapping, we can consider what the input *does* - how pressing buttons or joystick position moves the character through space. Here we worry not just about the choice of inputs, but also what kind of movement that generates. A few stereotypical examples are just a highlight of the huge variety of choices:

- In platformers like *Super Mario Bros.,* pressing left or right buttons moves the avatar by setting *continuous velocity* while the buttons are pressed, and resetting to zero when released. Some games like Donkey Kong Jr do it instantaneously, while Super Mario changes velocity with a very slight ramp up or down. Jumping also happens with the push of a button, but in comparison, jump is a *one-shot impulse* - it produces vertical velocity but only during the short transition from button up to button down.

- In games like *Asteroids,* pressing rotate buttons rotates the ship, by setting continuous *rotational velocity*. However, pressing the forward / backward buttons sets thrust, or *forward acceleration*, not velocity. The spaceship will slowly gain or lose velocity over time, as the player accelerates or decelerates. This makes the spaceship difficult to control (and deliberately so).

- In modern PC first-person shooters, so-called "mouselook" means that when the player moves their mouse, that is what the character will be looking at. In other words, moving the mouse changes the *rotational position* of the player's camera. In console shooters, however, players do not have a positional mouse, just analog sticks with limited sensitivity, so stick position typically maps to *rotational velocity* instead. Unfortunately, controlling camera via velocity is slower and less accurate than controlling position directly, so console games tend to compensate in some ways, such as by using *auto-look* and *auto-aim.*



(Examples of different mappings from button presses to avatar velocity, after Swink (TODO REF)).

These control mechanics are sometimes called *game feel mechanics* (TODO REF Swink), since they are all about the large variety of options in how player input maps to in-game actions, and how their ease or difficulty feels to the player. See the Further Reading section for more resources on game feel.

Progression mechanics

Another question we have to consider is: how will the player know how well they are doing? If they do not know whether they are winning or losing, doing better or falling behind, they might not enjoy playing the game – they will not be getting feedback that what they're doing has an effect.

In some games this question is hard to answer: in go or chess, players may not know exactly how they are doing until the endgame is at hand. Given this, players will try to use various approximations, such as counting how many pieces they have left and what positions they occupy, and converting that into a numerical score. On the other hand, other games provide very clear indication of player's progression, such as arcade games that show *current score* and *current lives* in the HUD.

Progression mechanics are the rules and elements for giving the player feedback on how well they are doing and how they are progressing, hence the name. There is a useful distinction between *overt* and *hidden* progression mechanics.

1. *Overt progression mechanics* show the player directly how well they are doing. Often this involves creating some explicit metric that is tied to player's performance and displaying it. Here are some examples of traditional ones:

    - *Score points* or *experience points* (XP) which is just a number that increases as the player accomplishes various actions in the game, shoots enemies, and so on.
    - *Levels*, which get earned for reaching specific experience milestones.

- *Achievements*, which get earned for reaching specific, typically uncommon goals.
- *Leaderboards*, which shows how players' scores stack up against each other.

2. *Hidden progression mechanics* change up the game in response to the player's progression. Very often, hidden progression mechanics are environmental changes: in a hack and slash game like *Diablo,* as you progress through the game you discover new territories that look very different, encounter new enemies, or pick up increasingly better loot.

   Another example might be growing the player's action space in response to their progress. In a life game like *The Sims,* the sense of "doing well" is told through the changing material situation of the family: being able to afford better furniture, moving to a larger house, or ultimately not having to work at all.

   Additionally, many aspects of the game can be treated by players as proxies for how well they are doing in the game, and take on the role of progression mechanics as well: for example, "how much money do I have" or "how large is my army" are easy for the player to treat as proxies for progress.

Whether overt or hidden, progression mechanics provide important feedback to the player on how well they are doing. In effect, they also encourage the player to keep playing. Scores and levels can be used to unlock more content or items in the game, in effect giving the player something to look forward to next. Unfolding story can also drive player to play more to see how it develops. Leaderboards and achievements appeal to their competitive and completionist natures, respectively.

The motivational aspect of progression mechanics is not cynically manipulative, it's just hard to imagine a game that keeps being interesting without giving the player feedback on how they are doing. But they must be added explicitly, and they must fit the context and theme of the game, otherwise players will try to infer progression mechanic via proxy, which might not be best for their experience.

Quick aside on gamification

"Gamification" is a recent term used to describe using various kinds of progression mechanics, typically explicit ones, to drive behavior outside of games. For example, the Q&A website Stack Overflow gives out XP and achievements to users who are recognized by the community as helpful, which unlocks some new actions, while Facebook and Twitter let people vote on their friends' posts with "likes", "favorites" and by sharing posts. These products are attempting to use game-like mechanics, in order to drive people to engage more with the product. However, *just having progression mechanics does not by itself motivate the player,* it can merely act as a multiplier on already existing motivation.

And so, gamification is precariously balanced: is can easily fail when the player is not already motivated to participate in the activity in the first place. Progression mechanics only give the player information about how they are doing, but if the activity is not interesting in the first place, they will not be able to make it interesting. We will have more to say about player motivation in (TODO REF CH5).

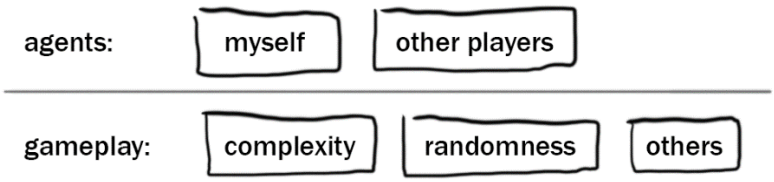
Uncertainty Mechanics

Games are more fun when the outcome is uncertain – but where does this uncertainty come from? In a pure gambling game like roulette, it will be down to pure luck of the random number generator. But in a

game like poker, there is some randomness, but also some strategy and psychology. Then in a game like chess, there is zero randomness, yet it challenges players' mental abilities to the point where their outcome will feel quite uncertain. For another example, a physical sport similarly challenges players with uncertain outcomes, but the uncertainty is based on their physical abilities instead.

Uncertainty mechanics are the techniques and solutions for adding interesting uncertainty to games. They have received a lot of attention (Costikyan REF), since they are fundamental across many game genres. Overcoming, minimizing, or predicting sources of uncertainty is an important part of mastery in many kinds of games.

This family encompasses the variety of mechanics centered on not knowing what will happen next, how that challenges the player, and how the player learns to manage them or cope with them in order to win. Here we follow Costikyan's taxonomy, which enumerates the following major and minor types of uncertainty.



(Types of uncertainty, after Costikyan (TODO REF))

1. *Randomness* – dealing with an unpredictable random process, like dice or a deck of cards.

   Knowing your odds at the blackjack table, or figuring out where you might land with the next dice roll in Monopoly, are some examples of why reasoning about randomness is crucial. Two types are worth considering:

   a) *Stationary processes*, where probability doesn't change from one sample to the next. Dice are a good example: past dice rolls do not affect probabilities in a future roll.
   b) *Non-stationary processes*, where previous samples change the probability of a future one. A simple example is a deck of cards. As we draw cards from the deck, the probability of an un-seen card being next increases, while the probability of already-drawn card decreases to zero (which is also the reason behind card-counting in gambling).

   Randomness is a very common kind of uncertainty, easy to introduce into various game rules, and an easy way to introduce dramatic tension and "shake up" predictable gameplay. However, it's also easy to overdo, as we discuss in (TODO REF CH?).

2. *Skill uncertainty* – not being able to do things, and training or learning to overcome it.

   Inability to do things is big source of uncertainty and negative emotions for new players, but it decreases as they learn how to play the game. Overcoming skill uncertainty is often required to

attain mastery at a particular game or sport.

a) *Performance uncertainty,* where the player has difficulty performing the tasks with skill or precision. Action games or sports, when the player is new to them, are a good example.
b) *Perception uncertainty,* where the player has difficulty seeing or sensing what they need. Games with overwhelming or subtle displays, such as RTSes or hidden objects games, are examples.
c) *Solver's uncertainty,* where the player has difficulty figuring out the right solution to the problem at hand. Puzzle games, whether casual crosswords or more involved adventure games, exceed at this kind of uncertainty.

Since the player's own ability is the source of uncertainty, we can make increase or decrease it by changing the game's difficulty, or letting the player train and adjusting the learning curve.

3. *Player unpredictability* – not knowing what your opponents will do.

This is a significant source of uncertainty in multiplayer games, since the player does not know what the enemies or collaborators will do, and learning to figure them out is a major challenge and a strategic advantage. This is present in a variety of multiplayer games as well as sports.

Player unpredictability also applies when the other player is a) a computer-controlled AI character, or even b) a computer-controlled game system (e.g. the game world, or the game economy). The challenge is similar: the player is challenged to figure out how the AI works in order to improve their chances of success.

4. *Complexity* – dealing with a complex situation that's hard reason about.

This kind of uncertainty is common in strategy games: chess, war games, management games, and others along these lines. We can distinguish two types:

a) *Analytic complexity,* where reasoning about the current game state taxes the player's mental abilities. This is very common for games with large action spaces and states with high branching factor, such as chess or war games, or games that exhibit chaotic behavior (TODO REF CH4).
b) *Hidden information,* where the player is missing all the data they need to make a good decision, such as in strategy games that employ fog-of-war, or even games like poker.

Analytically complex games tax the player's ability to infer the future from the present, and this kind of uncertainty can be improved by letting the players experiment and explore, and slowly build up a mental model of how the world works.

5. *Anticipation* - trying to anticipate what happens next in the overall game.

Sometimes the uncertainty comes simply from not knowing what the designer has in store for you. With this kind of uncertainty, players will be challenge to prepare themselves for whatever might happen next at the whim of the designer. We can distinguish three subtypes:

a) *Narrative anticipation,* where the player doesn't know what comes next in the story, so their ability to prepare will be challenged. This is particularly common in story-driven games where the player's trajectory through the game is deliberately full of interesting turns.
b) *Game evolution,* where the game simply changes over time, and challenges players to prepare. This is common with games such as Magic the Gathering, or social games such as CityVille, as well as esports games like Hearthstone, where the game is guarantee to evolve, and players who do not prepare will be left in the dust.
c) *Real life uncertainty,* which challenges players to adjust their real life schedules to the whims of the game in order to advance and win. This usually involves the "metagame" (TODO REF CH6), and is common in games with significant multiplayer elements, such as MMOs, esports, or other competitive multiplayer.

Players often enjoy anticipating the future, and then seeing how their predictions turned out, and if they were able to prepare accordingly. This level of uncertainty can be modulated by keeping the changes reasonable within the logic of the game universe, and giving players appropriate amount of clues about how things are going to change over time.

The various types of uncertainty are used to challenge the player, and make the challenge enjoyable. Games often employ multiple types of uncertainty at the same time: for example, a CCG like *Magic the Gathering* might interleave non-stationary random processes (cards) and hidden information, with player unpredictability, high amounts of analytic complexity (arising from the complex combat and resource mechanics) as well as significant game evolution and real-life uncertainty elements.

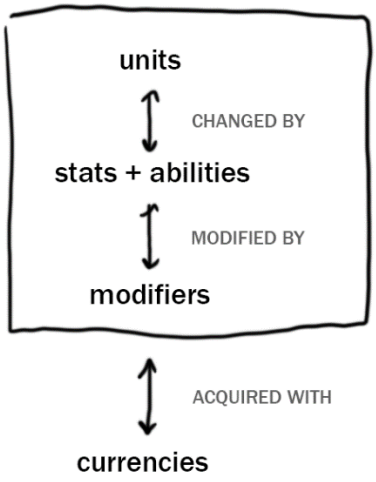
However, just like "enjoyment", the player's feeling of challenge is also subjective and based on their previous experience. Children like to play Tic Tac Toe because it challenges them: their ability to analytically predict how it will unfold is relatively low. But for adults, the game holds no secrets - and without that uncertainty, the challenge is also gone. A game that becomes predictable and rote also loses a lot of its appeal.

In the end, uncertainty is so common in games, it feels like a necessary mechanic. It seems virtually impossible to imagine a game that doesn't rely on uncertainty in some form or another.

Resource Management Mechanics

Another common family of mechanics concerns what the player has at their disposal to win the game. For example, they might control units in the game: armies, or vehicles, or spaceships – or maybe individual people, such as in role-playing games. The player could also own tangible resources like money or raw materials, to buy things, produce items, and so on – or perhaps own factories or cities or entire planets, depending entirely on the game.

We will use the name *resources* to denote the things we can acquire and use to accomplish our goals in the game. Combined with rules for how they can be management, these form the *resource management mechanics* family. Let's look at the variety of different ones we could consider.



(Types of resource management elements and their interactions)

1. *Units.* There are items in the game world that the player directs, manages, or perhaps simply encounters. Units are an exceedingly common resource mechanic. Game pieces in chess or checkers are examples of units. In computer games, even very early games like Space Invaders already had a variety of enemy units with different behaviors.

   Unit mechanics reflect questions such as:
   - What are the units in the game, and how are they controlled by the player or AI?
   - How do those units act? What are their abilities?
   - How can those abilities change, or be strategically improved?

   Every game has its own, unique design for units. However, games in specific genres often tend to cluster around common patterns:

   - *Classic war games* may have units that correspond to real-world military units, with similar movement and combat characteristics, such as some notion of firing range or movement range based on terrain.
   - *Real-time strategy games* like StarCraft have a number of unit types with very different, strongly differentiated abilities, interlocking so that players need a team of units that support each other. A global tech tree may unlock new unit types as the game progresses.
   - *Role-playing games* typically have fewer, more complex units. They are the characters that live in the game world, and they might have richer back-stories, stats, and extensive abilities which get unlocked as the game progresses.
   - *Management games* may have a few units that consume and produce specific combinations of resources, and they may even be non-moveable (like buildings in SimCity or cities in Civilization). The choice of what to build given the player's abilities and future threats presents trade-offs.
   - Some games have unexpected units: for example, player avatars in rhythm games like *Guitar Hero,* or the player's stable of cars in driving games like *Forza.*

- … and so on. The choice of units is going to very strongly describe what the game is about, and what kind of gameplay it will produce.

Genres also evolve typical ways for how those units will be controlled. Since the player usually has to deal with multiple units at once, UI issues become important: how does the player select units, group units together, issue commands, inspect their state, and so on.

2. *Abilities*. We have mentioned abilities in passing: they are the types of actions that units can perform. Some games keep the ability vocabulary limited: in checkers, each game piece follows very simple rules. But other games relish in introducing a zoo of units and abilities, such as role-playing games with complicated magic systems, or card games with card-specific abilities.

   The more abilities units can have, the more it opens up the player's action space, and challenges the player's ability to pick the best action. Complexity does not arise just from their number, it can also arise from their variety and how they interlock with each other: such as in chess, where a few, highly differentiated abilities produce a very large game state space.

   In some games, it is also possible to change or improve abilities over time. This is more common in computer than physical games, and can be used to present the player with interesting challenges. We can consider a player in an action-strategy RTS game, who just developed a well-balanced army of attack units, defense units, healer units, etc. Then during the game session, they research an upgrade that will make their attack units twice as fast, but less than half as strong. This presents a dilemma: does it make sense to upgrade all attack units? What will that do to the team structure, does this mean adding more healers or defenders, and is that even affordable? Often there is only one way to figure this out: try it, experiment, and see what happens.

   As a quick aside: designers differ on where they land on *minimalism vs maximalism*. Is it better to have more or fewer abilities, and more or fewer unit types? This is a very subjective question, and will present players with different challenges. On one end, some players relish games that require an encyclopedic knowledge of a large number of units and abilities, while on the other end, other players enjoy minimalism and sticking to very few rules that combine in complex ways. The right choice will very much depend on the genre, what kind of a player is going to play this game, and what player experience the designer is looking for.

3. *Stats*. When we're designing units with different abilities, and especially if we want units to change their abilities over time, we may want to quantify it in some way that's easy for the player to understand (and the computer to calculate).

   Numerical stats are a popular solution. They have been a part of games for a long time: early war games already used attack points, armor points, and movement points, to quantify how the different types of vehicles and human groups would behave on the battlefield (Dunnigan REF TODO). Not long after them, fantasy RPGs like AD&D inherited this approach, and started developing complicated character sheets with stats such as strength, dexterity, wisdom, and so on,

to describe the character's state. We see echoes of these early data models to this day in RPGs and other genres.

Stats are often used to:

- Modulate abilities, e.g. movement points specify how fast a unit can move per turn, a higher attack range means hitting targets further away
- Unlock abilities, e.g. higher strength let a fighter equip a secondary weapon with new abilities
- Resolve conflict or interactions, e.g. higher attack points mean more damage to the target
- … and so on

Changing unit abilities (or introducing new units with new abilities) changes up how the player plays the game, and forces them to discover new tactics and new strategies for winning. This is typically a good thing - it prevents the player from sticking to a single strategy that works for them.

4. *Modifiers*. Given a stat such as "strength", players may want to be able to improve this stat, and make the unit more powerful. Stat modification mechanics are called *modifiers,* or *buffs,* or *power-ups*. The idea is to reify the change in stats into a concrete in-game resource that can be acquired, and then strategically used by the player to modify their abilities.

   Games of all genres are full of stat modifiers:

   - In *Mario Kart,* collecting mushrooms gives you an automatic speed boost, while banana peels throw you off your trajectory, both for a short period of time
   - In *Diablo,* enchanted weapons or clothing will give boosts to various stats of the person wearing it, but there are many restrictions on who can wear what kinds of equipment
   - In *Civilization,* various technologies give a permanent boost to resource production from farms, mines, and other types of land tiles, but they are expensive to acquire
   - … and so on

   Improving stats (from the player's perspective) is often called *buffing* while reducing stats is called *nerfing*. Both buffing and nerfing accomplish the goal of changing up the player's abilities, but they have a different emotional effect: players prefer to see their units get better and more powerful, it feels like a reward for hard work, while having them nerfed feels like punishment for not doing something right. For this reason it is better to buff stats rather than nerf them, when tuning a game.

   Finally, even if a game has stats, being able to modify them is not strictly necessary. Games like *Space Invaders* and chess have units with a variety of abilities that don't change, yet their variety of abilities combine together to form challenging gameplay.

5. *Currencies and other resources*. We talked a lot about units and their abilities and stats. Units in a game are a kind of resources for sure, but there are other types as well:

   - *Money* and other *currencies*, which you can use to buy other resources, and perhaps gain by selling resources or as rewards for progressing through the game.

- *Items* that can be used, such as weapons or upgrades in war games, or things like clothing pieces or various collectibles in RPGs. You can store items in an inventory, or potentially equip them on your units.
- *Consumables*, such as fuel for your tanks or food for your soldiers, which will consumed in the process of buffing or enabling some other elements.
- *Raw resources,* such as crafting items, which are not usable directly but can be used to create other units (e.g. using steel to build tanks), or to craft items (e.g. turn iron and coal into steel).
- *Space* itself can be a resource, such as when the terrain you occupy produces a certain yield of other resources per unit time.

These kinds of resources are intended to be gained, exchanged, and perhaps consumed, and ideally they will interlock with each other and with other mechanics, and create an in-game economy. We will come back to economies, currencies, and crafting in (TODO REF CH4), when we talk about game systems.

There are more mechanics we could talk about in this section, although many of them would start getting fairly genre-specific: beyond some shared basics, the mechanics of racing game are going to be different from a turn-based strategy game, and different still from a real-time first-person shooter.

Mechanics design

When we create a new game, we will rarely create brand new mechanics that have never been seen before. More likely we will revisit existing mechanics, or their variants, and remix them and put them together in novel ways. But even this can be a tough design challenge, to redesign existing mechanics to make them fit our particular game.

We need our game mechanics to be interesting, so that the player will enjoy interacting with them. This is a context-specific design problem, and many bits of advice will be applicable only to some genres or games.

But fortunately, there are also a few more generic *heuristics,* or broad mental models, for how we could improve mechanics in general, and make them more interesting independently of context.

Heuristics

In his book *Aesthetics of Play,* Upton (TODO REF) offers six useful heuristics for how we can evaluate and improve game mechanics.[4] The heuristics are:

1. *Choice.* Player should perceive a range of possible actions.

   Players will have an immediate reaction to their perceived action space: if it is too narrow or constraining, they will get bored doing the same thing over and over again, but if it offers too many

[4] While Upton speaks of them in terms of action and intent horizons, this text adapts them using the vocabulary we have been developing so far in this text.

choices, they may get confused and frustrated. The ideal size of the perceived action space is going to depend on the audience, game type, player expectations, and so on (for example, compare expectations for a children's game like *Snakes and Ladders*, to a more sophisticated game like *Magic the Gathering*).

In practice, a simple rule of thumb is that games with more choices will often feel more open and rich than variants with fewer choices.

2. *Variety*. Actions aren't repeated.

Actions should be numerous, but it feels interesting to have the available actions *change* over time, compared to choosing from the same actions over and over. When new actions suddenly become available, or our favorite ones become unavailable, or when we get a whole new set of abilities we did not have before, it immediately grabs the player's attention. It forces the player out of a comfortable rut and into actively evaluating their new situation.

This is especially important because players will actively learn how to use existing mechanics to their advantage, so having the same unchanging mechanics will make it easier to master them, which in turn will make them feel uninteresting and "played out". Changing the set of mechanics over time is a good way to keep the player actively learning.

3. *Consequence*. Actions have outcomes.

This is a bit of a truism, but the decision to do or not do something should have a consequence, which the player can attribute to their decision. If the player's actions have no consequence (for example, whatever is destined to happen will happen), the player will lose the feeling of agency or control over their situation. And if the player's actions have a consequence that cannot be attributed (such as when the outcomes happens much later), it will prevent them from understanding how the game universe works.

Mastering consequence and causality in the game world is necessary for forming successful long-term plans and strategies. It is interesting for the player to figure out which actions have the best consequences, and juggle multiple opportunities whose consequences conflict with each other (such as the choice between short-term or long-term gains).

4. *Predictability*. Outcomes can be anticipated.

Assuming that actions have outcomes, player should be able to learn to anticipate those outcomes, at least to some degree. Again, being able to make theories about how the world works is at the heart of being able to make long-term plans. If the behavior outcome of every monster encounter was always completely random, it would not make the game interesting for very long.

Complete predictability and unpredictability are both undesirable, the most interesting mechanics lie somewhere in the middle. The player should be able to figure out how the various mechanics

work with enough effort, and then make use of them.

5. *Uncertainty.* Outcomes aren't predetermined.

   This is the flip side of predictability. A fully predictable outcome is a problem, just like a fully unpredictable one. The game benefits from a degree of understandable uncertainty, which the player may try to master.

   One common source of uncertainty are randomness generators, such as dice or decks of cards. However, a more popular source is other players, because they are both far more interesting, understandable, and also unpredictable, than simple dice or cards. Even in purely deterministic games such as chess, the presence of the other player fills each game moment with rich uncertainty.

6. *Satisfaction.* Desirable outcomes are attainable.

   The contrast here is between desirable and undesirable outcomes. If a game is completely unwinnable, if the player keeps failing and failing at what they do, they will quickly get frustrated and lose interest, and we must prevent that. Overwhelming difficulty is a common source of this kind of a frustration.

   At the same time, if the game always delivers desirable outcomes, if the player always gets what they want, this is not going to be interesting, either. We need to strike a good balance between challenging the player in interesting ways, yet making it possible for them to reach their goals.

These heuristics are useful, because they point out some general principles behind what makes mechanics interesting and engaging for players. We can keep them in the back of the mind as we seek to design new game elements and actions ourselves.

Primary and Derived Mechanics

Beyond designing the elements and actions of the game, we should also consider that players will generate their own abstractions on top of the ones we provide, as they learn to master the game.

For example, when we learn how to play chess, we learn about the pieces, the actions, and the rules of the game. We learn how to move the queen or the rook, we learn that pawns advance one or two tiles forward but attack on the diagonal, and so on.

But in order to play chess *well,* we need to figure out a slightly different set of abstractions. We need to learn the concept of protecting the king, and how to gain control over the game board, what it means to sacrifice, how to fork, or how to pin. We may want to learn about endgames, and acquire an encyclopedic knowledge of standard openings. And this is by no means limited to chess – competitive Hearthstone players have their vocabulary of decks and deck types, FPS deathmatch players talk a lot about level tactics, camping or kill stealing, and so on.

This is the difference between primary mechanics and derived ones. *Primary mechanics* are what is required to play the game, they define the gameplay. *Derived mechanics,* on the other hand, are player abstractions that have been created and defined by the community of players, usually distilling extensive experience playing the game and noticing patterns. In some games, they may also be called *standard tactics* or, less often, *second-order mechanics*.[5]

This kind of expert mental modeling is not limited to games – experts in all sorts of areas will commonly identify patterns and structures that occur repeatedly. Derived mechanics are not required to play, but they are very useful since they encode expert knowledge about how the game is structured beyond the obvious ways.

As designers, we are not obligated to design or predict higher-order mechanics. We may not even know what they are until the player community discovers them. However, as we design and playtest the game, we need to be on a lookout for them, in case that players discover some mechanics that make the game too easy to figure out, or otherwise not enjoyable.

Also, as a side note, the existence of derived mechanics points out a tension in our understanding of the word "mechanic": if regular mechanics are the game objects and actions, should these higher-order abstractions also be called "mechanics"? Perhaps not, but at this point we do not have a better name for them, although *standard tactics* is a good substitute for many games.

Further reading

Taxonomies of mechanics:

- "Patterns in Game Design" by Bjork and Holopainen
- "The 400 Project" by Hal Barwood and Noah Falstein, an unfinished project to document 400 rules of game design, http://www.theinspiracy.com/

Discussion of Mechanics:

The Aesthetic of Play by Upton ch. 3-4 discuss player's action spaces, the distinction between actual and perceived actions, and the heuristics for designing them.

Game Feel by Swink contains a highly detailed examination of action mechanics, including great timing curves of how mouse inputs maps to in-game actions over time.

- *Uncertainty in Games* by Costikyan provides a thorough examination of types of uncertainty and how they are employed, using many examples from existing games

[5] This division into first order and second order mechanics is present in (Elias REF, and Church REF) among others. However, cardinal ordering only invites questions we do not know how to answer: for example, if we have first and second, what could be third-order mechanics? Instead of pursuing this further, we will only differentiate between primary mechanics, and all other ones.

- For details on multiplayer mechanics, including a large variety of decision points for multiplayer game designers especially of board and card games, *Characteristics of Games* by Elias et al is highly recommended.

- For a history of unit and resource mechanics in the context of board wargames, see the incomparable *Wargames Handbook, Third Edition* by Dunnigan.