What I've learned about designing multiplayer games so far

by [Daniel Cook](http://www.gamasutra.com/blogs/author/DanielCook/4054/) on 01/04/14 03:32:00 pm

***The following blog post, unless otherwise noted, was written by a member of Gamasutra’s community.***

***The thoughts and opinions expressed are those of the writer and not Gamasutra or its parent company.***

*This post originally appeared on my blog* [*Lostgarden.com*](http://www.lostgarden.com/2013/12/multiplayer-logistics.html)*. You can follow my vitriolic tweets about Ian Bogost's latest facial hair misteps via* [*@danctheduck*](https://twitter.com/danctheduck) *on Twitter.*

How do we get players to play together in a manner that fits their schedules? This is a key logistical challenge a designer faces when building multiplayer games.

## The promise

We are seeing a blossoming of innovative multiplayer systems. In previous eras there were a handful of default models that games might use (matches, play-by-mail). Games today exist on a spectrum from fully concurrent to fully asynchronous and everything in between. A game like Dark Souls is predominantly single player, but includes interactions that are asynchronous (the leaving of messages and deaths) or fully concurrent (the joining of another player into your game for PvP or Coop.)

We are entering a golden era of multiplayer gameplay. Server costs are falling dramatically with the advent of cloud computing. Broadband internet and always on mobile connections are spreading rapidly across the globe. Business models like in game payments, crowd funding and service-based gaming are evolving to the point to financially support a broad range of long-lived communities. Designers are playing with these new capabilities to invent new forms of multiplayer gaming.

## The challenge

However, multiplayer is both expensive to build and has a high risk of failure. Often teams invest 50 to 100% of their development budget into creating a multiplayer mode. It seems worth it. During development, the team plays every Friday and has so much fun they are convinced that multiplayer is what will turn their game into the next League of Legends or Counter Strike.

The real test occurs when the game faces a live population of players. Upon launch, multiplayer games often see only a few weeks of active multiplayer activity. Too many people show up. Then not enough. Players visit sporadically and the player experience is deemed unreliable. The active matches trickle down to nothing. The traditional matchmaking lobbies (a design from the 1990’s) are left empty and will never be full ever again. The multiplayer portion of the game dies a sad sputtering death.

I see this as a challenge of logistics. There were players who wanted to play. However the way that the game put those players together results in weak community that was unable to self sustain.

Are there atomic elements of multiplayer logistics that lets us approach the topic of inventing new systems in a more rigorous fashion? Simply copying multiplayer patterns from previous eras works poorly. To invent new multiplayer modes, we must have conceptual tools that let us clearly and concisely manipulate topics like logistics, concurrency and interaction schedules.

**Concepts when talking about multiplayer**

Here are some concepts I think about when designing a multiplayer game.

## Interactions

[[](https://3.bp.blogspot.com/-ZXlXubivRg0/UsMfRWbgpxI/AAAAAAAABAY/6HNo_tBMRjg/s1600/Multiplayer_Interaction+Basics.png)](https://3.bp.blogspot.com/-ZXlXubivRg0/UsMfRWbgpxI/AAAAAAAABAY/6HNo_tBMRjg/s1600/Multiplayer_Interaction+Basics.png)

You can break up any multiplayer system into a series of interactions. An interaction is anytime players interact with one another via a game system (be it chat, hitting one another, etc.) These are the multiplayer verbs of your game. Usually a game has a set of single player verbs (move, quit, etc) and another set of multiplayer interactions mixed in. Interactions have a wide range of multiplayer properties such as frequency, scope, mode, etc.

If you map an interaction onto time, it looks something like this

* The player starts the interaction
* They end the interaction
* They wait for a response.
* If no response is forthcoming, they leave.

Interactions aren’t a new thing. The structure is identical to that found in atomic game loops. However, instead of a single loop you have something closer to a figure 8 with at least two participants. These concepts go back to communication theory that [Chris Crawford](http://www.amazon.com/The-Art-Computer-Game-Design-ebook/dp/B0052QA5WU) adapted to games design theory in the 1980’s. This is fundamental stuff that all professional game designers should know.

[[](https://4.bp.blogspot.com/-bFjTSY-AqQQ/UsMfRnE9jyI/AAAAAAAABAk/TKA99px2Z04/s1600/Multiplayer_Reciprocation.png)](https://4.bp.blogspot.com/-bFjTSY-AqQQ/UsMfRnE9jyI/AAAAAAAABAk/TKA99px2Z04/s1600/Multiplayer_Reciprocation.png)

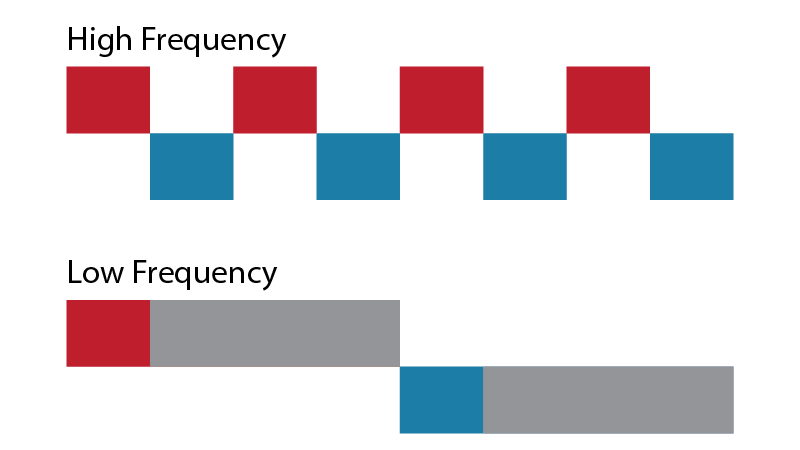
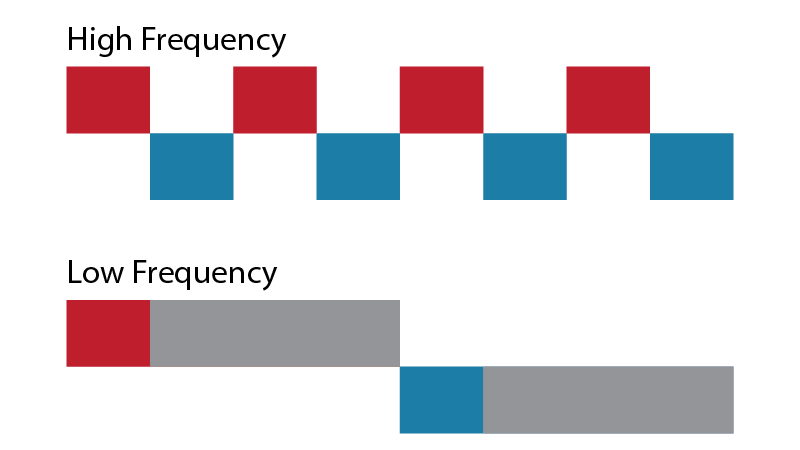
Initial loop:

* Model A: Player formulates an action and a target player or group.
* Action A: Player performs the action.
* Rules: The results of the action are mediated by the game logic.
* Response A: Player A sees the immediate results as generated by the game.
* Response B: Player B sees the immediate results as generated by the game. Note that what Player B sees is likely different than what occurs for player A. This naturally leads to divergent mental models and enables gameplay concepts such as hidden information or Yomi.

Reciprocating loop

* Model, Action, Rules, Response B: The target players tries to understand what happened and formulates a response.
* From here the loop ping pongs back and forth between participants.

## Frequency of interaction

[[](https://4.bp.blogspot.com/-Qp6TNrzi1ds/UsMfSNg_keI/AAAAAAAABA8/247zB0UtaJA/s1600/Multiplayer_frequency.png)](https://4.bp.blogspot.com/-Qp6TNrzi1ds/UsMfSNg_keI/AAAAAAAABA8/247zB0UtaJA/s1600/Multiplayer_frequency.png)

What is the frequency of interaction necessary to yield the impression of concurrency? You may find that you need to interact once every 5 minutes in a strategic game like Civilization while you need to interact every 200 ms to create the same impression in a twitch-based action game like Counter-Strike. See the article “Loops and Arcs” for a more detailed explanation.

In general, the higher the frequency of interactions, the more information being communicated between players. This can increase the pace of relationship formation.

As with many interaction variables, there are distinct phase changes in the players perception as the frequency hits a threshold. Simply by changing the spacing between interactions, we get radically different forms of play (and associated logistical challenges):

[[](https://4.bp.blogspot.com/-Z808ASB1toU/UsMfSpF-QZI/AAAAAAAABA4/fxXb-ocFwoQ/s1600/Multiplayer_realtimevsasync.png)](https://4.bp.blogspot.com/-Z808ASB1toU/UsMfSpF-QZI/AAAAAAAABA4/fxXb-ocFwoQ/s1600/Multiplayer_realtimevsasync.png)

* **Real time:** Players perceive interactions as ‘real-time’ when the frequency reaches the point where: A player starts and ends an interaction and then sees a response before they move onto other tasks; interactions overlap. Chat, for example, can feel real-time despite there often being more than a minute between responses. Real-time systems have less need for persistence but are often more expensive to run and build.
* **Asynchronous interactions:** The frequency at which a player can start an interaction and end the interaction and then quit the game without seeing a response is seen as asynchronous. Generally you build in some sort of persistence so that a player that logs in later can see the results of the interaction and formulate a response.

## Types of interaction

There are a variety of interaction types. Think of these as ‘how’ players interact. For a much more in depth description of all the various multiplayer interactions, see Raph Koster's seminal talk on [social game mechanics](http://www.raphkoster.com/wp-content/uploads/2011/02/Koster_Social_Social-mechanics_GDC2011.pdf).

* **Spacial avatar interaction**: Two or more avatars interact with one another. Shooting players in Quake is the classic example. Following a player in Journey is another.
* **Spacial environment interaction**: Players also interact through the intermediate environment. In Minecraft, players build castles that other players then explore. For a higher frequency example, in Bomberman, players place bombs that open up passages or do damage to others.
* **Decoration and Display**: Players signal status, affiliations and history via what they wear or how they decorate their weapons, pets and houses.
* **Economic**: Players give, trade or pay for various resources to transform or transfer to another player. This can be a typical sale of a sword to another player for gold. Or it can paying mana for a buff that boost the health of a nearby player. See Joris Dormans work on internal economies for more on this topic.
* **Text**: The most common method of introducing language into an online game is through text. It tends to be low cost and there’s a rich set of tools (spam filters, stylistic conventions) for dealing with common issues. It tends to work best with a keyboard.
* **Voice**: Voice offers additional nuance including emotions, age, gender and more. It has limits for group size, bandwidth and is notoriously weak when it comes to filtering.
* **Body language**: In local spaces like on a couch or around a table, we pick up on high bandwidth communication such as facial expression, posture, body height and physical presence. When a tall pretty boy looks you in the eye and asks that you trade your rare treasure with him, you may be getting signals that go far beyond what is found in other types of interaction. This creates rich emergent multiplayer gameplay. However, it is also hard to mediate and incorporate explicitly into the game systems.

## Size of community

There are also massive phase changes that occur as you increase the number of participants in a community.

* **1 player**: Mastery, progression, exploration, narrative are available as design tools.
* **2 players**: Communication, relationships, status, gifting, trade, cooperation and competition become available.
* **3-4 players**: Alliances, politics, gossip, othering/stereotyping become available.
* **Small group** (5+): Group vs group interactions, Official leadership, role specialization, official punishment
* **Medium group** (12+): Factions, barter economies, and banishment
* **Large groups** (40+): Hierarchy (leaders and sub-leaders), Currency-based economies, role enforcement. Adhoc systems of government, public codification of social norms.
* **Very Large groups** (200+): Merchant classes, market-based pricing, codified systems of government, underclasses, celebrity, propaganda. This is the point at which a players is guaranteed not to know everyone and official systems are required to make social norms work. (see [Dunbar's Number](http://en.wikipedia.org/wiki/Dunbar's_number))
* **Massive groups** (1,000+): Polling, city-scale production efforts. There are very few dynamics that happen at this scale that isn’t also explore with 200+ or even 40+ groups.

I'm defining these groups in the context of player interactions.  The actual game population may be much larger.  For example with trade in Realm of the Mad God, we saw simple trade interactions happen with as little as two people even in populations that are in the thousands.  Two good rule of thumb when considering group size is to ask:

* **Who does this action impact or target?**  This gives a rough estimate of the group size your system needs to support.
* **Is a larger group size necessary for this behavior to emerge?**  If not, you can usually get by by targeting your design at multiple instances of a smaller group size.

The actual transition points fluctuate around these numbers based off contextual factors. For example, the transition to the dynamics of a Very Large Group can occur as soon as 60 or 70 people if there are weak communication channels that stress a player’s ability to maintain relationships.

Also, large groups are inevitably composed of smaller groups. So as systems are added, the dynamics of lower number groups are still present.

### The dangers of large group sizes:

*It can be tempting to make epic multiplayer games with thousands of interacting players that could theoretically all fit in the same room. However, the technology and design costs are high and the benefits weak. Past 150-250 players, your game is in territory beyond Dunbar’s theorized biological limit on maintaining meaningful relationships.  All those extra people end up just being treated as number or abstractions by your players. A simple sim or polling system can often capture the major benefits of the next highest group size.*

*Realm of the Mad God was completely playable as an MMO with action sequences of 40-80 players and trade / hub interactions of ~150.  Players did not miss the 1000s of players.*

*This reality raises serious questions about the need for designs that emphasize ‘massively multiplayer’ experiences. Just because a concept sounds exciting (“a million people building a new society!”) doesn't mean it is a smart design. Human social capacities are limited and we can (and have!) over-engineer multiplayer systems.*

## Scope of interaction

How many people does a single interaction impact? A player can interact with a single individual or they can interact with one of the group sizes listed above.

* **Targeting a player interaction at small groups:** With smaller group sizes you get communication similar to a conversation. There is a clearly defined interaction loop that can stabilize on a set of shared vocabulary and social norms quickly.
* **Targeting a player interaction at larger groups**: With larger group sizes you see more broadcast scenarios and interactions are broader, less tailored to individuals. When interacting with large groups, it is common for the massive response to flood the recipient with too much information. Extreme reactions are also more common as people talk over and past one another.

## Degree of interaction

* **Parallel**: Players can behave independently from one another. A ghost racing car rarely impacts another player. Often the primary benefit here is a sense of presence though it can also tie into lower frequency zero sum interactions like a leaderboard.
* **Zero Sum**: The action of one player blocks or reduces the interaction of another player. In Habbo hotel, movement is a zero sum interaction since the placement of one character blocks another character from occupying the same spot. This was famously used as a griefing tactic to box in players.
* **Non-Zero Sum**: The action of one player benefits another player. In Realm of the Mad God, shooting an enemy makes that enemy easier to kill for other players. Killing an enemy gives XP to everyone on the screen.

## Matchmaking

Matchmaking is the computer mediated act of introducing players to one another so they might interact.

This is a very broad definition of matchmaking, but is useful in the context of the wide range of multiplayer systems available. For example, a traditional console title might match players together by requiring players in a shared lobby to manually join a specific game. In Realm of the Mad God, players notice groups of players on a shared map and teleport to them. Both are forms of matchmaking, but they appear quite different in the player’s mind.

You can treat matchmaking abstractly as another interaction with a wait time.

## Matchmaking window

[[](https://1.bp.blogspot.com/-dtJ9INg8Q3k/UsMfRdRzGHI/AAAAAAAABAs/nZtmw8PD91Q/s1600/Multiplayer_Interaction+matching.png)](https://1.bp.blogspot.com/-dtJ9INg8Q3k/UsMfRdRzGHI/AAAAAAAABAs/nZtmw8PD91Q/s1600/Multiplayer_Interaction+matching.png)

The time you have to introduce a player looking for a multiplayer experience to another player. If the window is too long (and the player is not entertained during the window), they will leave.

## Matchmaking failure

When a player comes online and there is not another player immediately online, the players will quickly become bored and leave. There is often an implicit promise of a fun multiplayer experience and if you don’t deliver that in seconds, your game is judged as a failure.

What can be frustrating to the developer is that another player pops in a minute later and experiences the same exact thing. If one players sticks around long enough, another player will show up.

***Calculating daily failure threshold****: If the matchmaking window is W in minutes, then failure will occur when the daily active population is less than Minutes In a Day / W. So for example if people are only willing to wait half a minute, you’d need a daily active population of 1440 / 0.5 or 2880 players. Actual results will be lumpy because we are dealing with a statistical process and player populations peak around specific times of day.*

*This may seem quite reasonable, but if you are matchmaking primarily with small groups of friends, players may feel like no one they know is ever on.*

## Fragmentation

When the player population is segmented by social groups, game modes, players skill levels, time playing and other factors, it becomes fragmented. This reduces the actual concurrent player numbers available to the matchmaking system and increases the chance of a matchmaking failure.

***Example of fragmentation****: Suppose a game has 3 multiplayer modes and matches players into 10 skill categories. If the daily failure threshold is 2880 (from the previous example), then in the worst case scenario, you’d need 3x10x2880 or 86,400 concurrent players for everyone to get their first choice.*

Fragmentation creeps into a design. Someone wants to add another event or another game mode. The code is free, so why not? Surely the players will self sort. They do a little, but mostly they wonder why the matchmaking experience is so painful and then leave your game in frustration. Avoid fragmentation creep and put players together in big easily matched buckets when possible.

## Concurrency ratio

Any game has a number of active accounts and a number of players that are online at once. Players cannot be playing constantly and are often offline For example, an MMO might have 100 active subscribers, but only 10 of those are on at any one time. This would result in a concurrency ratio of 10:1. Some typical concurrency ratios:

* **MMO**: 10:1
* **Online Console Service** (like Xbox Live): 25:1
* **Individual Console game**: 150:1
* **Flash game**: 250:1
* **Couch multiplayer**: 1000:1 (This is slightly facetious. Couch multiplayer games are often played a few times a year)

***The Active User Trap****: One common mistake is that developers assume that high active player numbers will result in robust multiplayer communities. However you really need to look at actual concurrent users since many game types have extreme concurrency ratios. A game may have 1000 players but when each of those logins last 5 minutes and are spread over a week, you’ll average 0.5 concurrent players. If your matchmaking system doesn’t deal well with these sporadic, tiny populations, the game dies.*

## Relationship strength

Not all player interactions are equal due to unique relationships between players. Players build complex social models of other players both in game and out of game. Strangers are understood through simple, stereotype-based models. Close friends are understood through complex individual models built up over thousand or millions of minute reciprocation sequences.

Building mental models of another human is a biologically expensive operation. We seem to be able to keep 5 to 9 detailed models active at any one time though we can store many more at various levels of detail. Friendship is rare, complicated and built over long periods of time.

There are numerous benefits and trade offs that come from gaming with strangers or friends and friend-based play is often highly desirable. Games can help create friends by promoted repeated positive interactions. The higher the frequency, the quicker the relationship evolves. Relationship strength is a spectrum, but there are two commonly drawn categories

* Multiplayer with Strangers
* Multiplayer with Friends

## Multiplayer with Strangers

Let’s tackle multiplayer between strangers online first.

Pros:

* Anyone playing the game can be matched with anyone else with little regard for existing social bonds.  This model becomes immensely attractive when there is a small initial player base. Often this means if 10 people are online, 10 people can be playing together.
* Strangers, particularly young males, historically tend to compete with one another. This means that player vs player games that emphasize open conflict are an easy means of generate fun for some stranger populations.

Cons:

* Strangers have weak bonds and will not naturally engage in prosocial activities like collaboration.
* Skill differentials matter since players tend to compete. This forces developers to focus on segregating experts from newbies and fragments the population.
* Not all player populations thrive on overtly competitive gameplay. Some players prefer to collaborate. Others compete quietly for status by manipulating social relationships. These are difficult in stranger scenarios.

## Multiplayer with Friends

Pros

* Players are much more likely to schedule time together to play.
* Cooperative and communication heavy activities are considered fun.
* Mentoring between divergent skill levels is more likely to occur.
* Competitive play is still valid.

Cons

* There’s often little overlap between existing social groups and interest in a specific game.
* There’s often little overlap between existing social groups and share scheduled.
* Friend groups are small. Engaged players typically have 5-9 close relationships. Casual acquaintances may be higher in number, but in practice may act more like strangers. If you have 10 friends and the concurrency ratio for a service is 25:1, you will essentially never stumble upon them online.

## Tools for dealing with multiplayer logistics

So far I’ve just talked about the concepts behind multiplayer. Now we’ll dig into some common patterns that make use of these. There are three broad architectures:

* Match-based games
* Room-based games
* Asynchronous games

# Tools: Match-based games

Due to the long history of event-based matches in sports and board games multiplayer computer games often are organized into matches that start at a specific time and stop at a specific time or win condition.

Matches are the default logistics model used for many console and PC-style online games. They are immensely problematic. The matchmaking interaction has a very narrow window during which it requires a full set of players to show up in order to enter the game successfully. If you don’t get in, you need to wait till the next match starts. If this time is longer than the wait window, you’ll quit. Considering concurrency ratios, fragmentation and the burden of a tiny matchmaking window, it is not surprising that only the most popular match-based online titles survive.

## Scheduled Events

Ask people to show up at the same time. This essentially shifts play times so that they are on at the same time. Scheduling is an expensive planning activity on the part of the player. You’ll get a low overall engagement rate but those who do participate are likely to find other to play with. A special Halloween boss encounter in a MMO is an example of a scheduled event.

Events can be scheduled by the game developers or they can be scheduled by the players. Player scheduled events have the benefit of stronger social ties in play. Folks that get together for a board game night are such an event. The downside is that arranging meeting is a convoluted process (as anyone that tries to set up meetings with more than 6 people can attest). It often requires leadership or persistence, attributes that are often in low supply for lightly engaged players.

## Regularly scheduled events

If you can make the event regular, people will get in the habit of being at a particular place at a particular time. This reduces the cost of planning for the player and they can just reliably show up at a specific time instead of worrying about conflicts. A standard Wednesday game night for a guild is an example of a regularly scheduled event.

## Short matches

If matches are short enough (2 minutes? 30 seconds?) players that don’t get into the current match wait less time than the matchmaking window and thus are still around when the next match starts. Online word games do this, but it could be readily applied to other titles.

## Spectating on matches while waiting

If you can keep players entertained by letting them watch the game in progress, you can lengthen the matchmaking window. Games like Counter Strike do this upon entrance into a server and upon death.  Chatting is often tossed into this mix since it is a nice downtime activity that can build relationships.

## Bots during matchmaking to fill waits

Instead of putting players in a queue where nothing happens, put them directly into a match with bots as the opponents.

Getting bots that act like humans is often a tricky Turing test to pass. Not letting players talk and having a very narrow window of expression helps. When players learn this is happening they will start to distrust the game and question if all opponents are bots.

## Mechanical Interdependencies

Create activities that require multiple people to show up in order to achieve success. Not showing up lets down the group and thus increases the social pressure to show up. This can take the form of explicit roles or by limiting resources so that players can’t accomplish large goals independently.

# Tool: Room-based games

Ultimately match based games result in often insurmountable logistical issues for smaller games. A favorite alternative is room based games. Unlike a match which has a distinct start and exit, room-based games create a persistent playspace that players may independently join the game in progress (or leave the game in progress)

Rooms have a maximum number of ‘slots’ or spaces for players to join them. Once the room is full, no more players may join. This dramatically reduces the load on matchmaking. All you need to do is find a room with an empty slot available and dump players into it.

The downsides to rooms is that they eliminate certain game types. Group starting times are obviously out which eliminates most traditional sports. Games with progression arcs result in players that start at different types having differing levels of progress. You need to get creative.

A game like Journey is essentially a room based game with join and leave in progress. The max slots was 2 and as long as there were two concurrent players you could have a multiplayer experience. Most MMO’s are room-based games with very large rooms.

## Join In Progress, Leave in Progress

One reason why rooms offer such improved logistics over strict matches is that players may join or leave at any time.  Since it is highly unlikely that everyone will leave at once, especially in games with a predominance of parallel interactions, shortly after one person leave another person will join and you'll get a consistent average population in the room.

Pure match-based games are often quite rare because many popular games treat the individual server as a room and the match-based elements are merely scoring atop a dynamic population of players joining and leaving in progress.

## Elastic Room Instances

Create and remove rooms to fit that maximum currency. Given a room of maximum size N, you create new rooms so that the number of rooms equals Concurrent Player / N. So if 10 players are online and your default room size is 4, you’ll make sure there are 3 rooms to join.

To collapse a room, just wait until it naturally empties out as players leave the game or kick people out due to some in-game event intended to free up the instance. Once the room is empty, delete it. By giving rooms priority, you can fill the highest priority rooms first and kill off the low priority rooms. The result is that almost all rooms are constantly full and only the remainder are left alone.

We used this when creating world shards in Realm of the Mad God. The world generally felt full even when the concurrent population fluctuated dramatically.

## Default to single player gameplay for rooms with one player

Room-based games have the ‘remainder’ issue. A given maximum room size rarely divides evenly into the concurrent population. If the room size is 2 and there are 3 players online, there will be 1 player placed in a new room by themselves.

To deal with this scenario, it helps to have a game that is playable as a single player game until the next player joins the room.

A retail game like Dark Souls assume very low concurrency and plays almost entirely as a single player game (with light async ghost interactions) The concurrent matchmaking is a silent parallel interaction that happens without interrupting the single player adventuring. Since having a second player in the right place at the right time is uncommon, the game instead treats it as a special occurrence. (Note that since Dark Souls promises a single player game, they make the concurrent multiplayer experience opt-in through the use of soapstones. The soapstones signal that a successful match has occurred and the player must accept it. Respect your initial promise when you mix single player and multiplayer interactions.)

# Tool: Asynchronous techniques

## Play-by-mail

A player complete an interaction and then the game signals to them that they have a very long period of time before the other player responds. The next day or so, the other player sees the first player’s action and composes their response. This can take place over days.

Words with Friends is a modern example of this technique, but the practice goes back decades if not centuries (if you include play-by-mail board games). It is an intimate method of play that works well with text communication much like instant messages or email. Play-by-mail is very amenable to play between friends.

A downside is that players are deeply impatient. A single turn may not be all that satisfying and then having to wait multiple days for a response has a major drop off in retention. There are still matchmaking issues if fragmentation is too high but the explicitly long wait window ensures players don’t get too worried that the system is broken (they may just not like the system). The other downside is that in turn-based games, the non-response of one player may block another player.

## High Capacity Play-by-mail

One solution is for a player to start a large number of play-by-mail games. Given a response time of T days and a desired average wait time of W days, then the optimal number of games going at once is T/W. (So if you want a game popping in every hour and it takes 24 hours to response, then you need 24 games going.)

One added benefit of all this is that player response times are semi-random. This acts as a random reinforcement schedule and can result in very long term retention.

The downside to the technique is that it requires players to start up a lot of games in order to reduce the wait window and motivating players to do so is tricky. Automated game matching may be an answer.

## Inviting

You can leverage active players to invite new players to the game. These players often have strong relationships with the player and can potentially act as a source of new players into the game.

## Match with friends

Since async forms of multiplayer rely heavily on players to come back later, their game designs often relies on social connections outside the game as a form of additional pressure. If you can get people to invite or match with friends (as in Farmville) a lack of reciprocation in interpreted as putting their existing relationships at risk. The threat of being rude or seeming like you don’t care to someone you like is often enough of an incentive to encourage returning to the game.

Systems that play off existing relationships run the risk of alienating players. Players not invested in the game tend to find mechanical interactions annoying. Authenticity and intentions matter when it comes to human relationships.

## Visiting

In building games, you may create a persistent structure such as a town that other players can then visit independently of your presence.

Clash of Clans uses this when players attack your town. The town is a persistent structure that then acts as a level for the other player to conquer.

Visiting usually boils down to a simple resource exchange despite the trapping of being something more meaningful. The issue comes from questions of what happens when multiple people visit at once and the solution is to spin up different instances.

Jason Rohrer’s The Castle Doctrine uses the unique design of making visiting a blocking interaction. This opens the possibility for permanent changes being made to the visited location. One can imagine more complex versions of musical chairs as the foundation for some innovative designs.

## Ghosts

Record players behaviors and then play them back alongside the player in a similar environment. This works particularly well with parallel interactions like you see in racing games. It can also work with the rare non-zero sum interactions like you see in multiple time track games like Cursor 10 or Super Time Force. Ghosts gives a sense of presence but removes the matchmaking time constraints.

The downside is that ghosts usually works poorly with blocking or zero-sum interactions. The other downside is that if the ghost data and the environment get out of sync, then the ghost data becomes invalid. These can be alleviated slightly by either skipping blocked actions or falling back on AI behaviors that manage exceptions

On a more abstract level, ghosts are just tracks of player data that can be replayed on any sort of trigger. They can be triggered at the start of a race, when the player comes onscreen or when the player uses the special amulet of Ally Summoning.

# General practices

This essay has covered a lot of ground (and is still incomplete!), but I’ll leave you with a few quick recommendations.

* Don’t fragment your matchmaking population. Be very wary of the point at which your concurrent game’s matchmaking fails due to high concurrency ratios.
* Use room-base methods where possible, not match-based play.
* Persistence is your friend since it enables asynchronous interactions.
* Relationships are your friend since they increase retention. Try to build them where possible.
* Prototype early and deal with low populations density issues during the prototyping phase.

**Conclusion**

I remain quite excited about new multiplayer games. When I look at the theoretical advances being made with game grammar via Joris Dormans internal economies and some of the multiplayer concepts in this essay, the unexplored space for new forms of game seems vast. If you want to make your mark on our modern world, make a great multiplayer game. Solve the logistical issues that prevent people from playing together and build a game that spread quickly and easily throughout communities.

take care,

Danc.