2014-06-05

Millions of Games per Hour

└─Outline

Outline

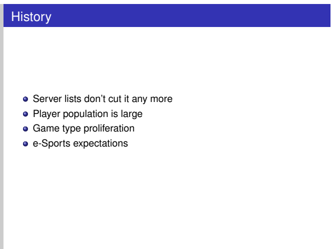
- brief intro about myself
- a bit about history and the problem space
- how Battle.net abstracts matching to deal with multiple titles
- Cooperative matchmaking in the context of Diablo 3
- Assigning games to available hardware
- Competitive matchmaking in the context of Hearthstone and SC2
- NEXT: history

2014-06-05

Millions of Games per Hour

└─History & Problem Space

└─History

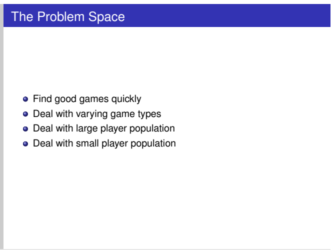

History

- Server lists don't cut it any more
- Player population is large
- Game type proliferation
- e-Sports expectations

- Server lists make unwieldy UI
- High population => long list of servers
- Large # of game types => deep menu choices
- Novice doesn't know where to go to get a good game
- Can encourage cliques
- e-Sports needs proper ratings
- NEXT: problem space

2014-06-05

Millions of Games per Hour
└History & Problem Space
└The Problem Space



The Problem Space

- Find good games quickly
- Deal with varying game types
- Deal with large player population
- Deal with small player population

- find good games in a timely manner
- legal: fulfils reqs
- quality: appropriate skill match (comp) / time left to play (coop)
- player population: game duty cycle
- small pop: sensible tradeoffs
- NEXT: more problem space

2014-06-05

Millions of Games per Hour
└History & Problem Space
└The Problem Space

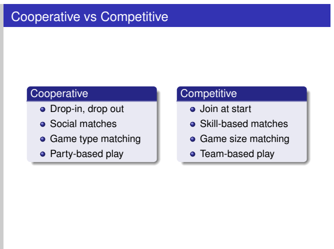
The Problem Space

- Assign games to servers
    - spread load evenly
    - fill up servers
    - deal with overload scenarios
- Deal with community preferences

- fixed server pool: spread evenly
- can spin up servers: fill them
- overloading requires queueing for timely games and fairness
- Battle.net supports more players than any one game
- flexibility to configure new scenarios according to emergent play
- NEXT: coop vs comp

2014-06-05

Millions of Games per Hour
└─History & Problem Space
└─Cooperative vs Competitive



Cooperative vs Competitive

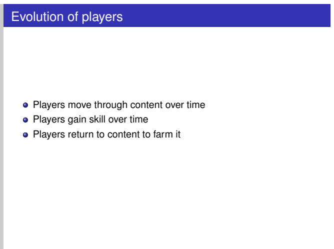
Cooperative
- Drop-in, drop out
- Social matches
- Game type matching
- Party-based play

Competitive
- Join at start
- Skill-based matches
- Game size matching
- Team-based play

- major difference is drop in/out vs join-at-start
- comp design/tech doesn't permit drop in (economy, game state)
- comp skills system doesn't permit drop in (binary outcome, no partials)
- coop requires drop in/out (social play)
- coop: partition games by type
- comp: partition games by size
- team based = party based
- NEXT: player evolution

2014-06-05

Millions of Games per Hour
└─History & Problem Space
└─Evolution of players



Evolution of players

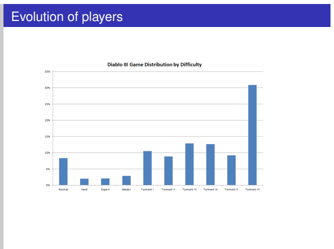
- Players move through content over time
- Players gain skill over time
- Players return to content to farm it

- day 1 load profile is different from day 100
- expect players to be on a bell curve
- parameters of bell curve change over time
- specific parts of the game will be sticky for farmers
- hard to predict ahead of time what the sticky parts will be (depends on design, may change)
- NEXT: graph

2014-06-05

Millions of Games per Hour
└History & Problem Space

└Evolution of players



Evolution of players

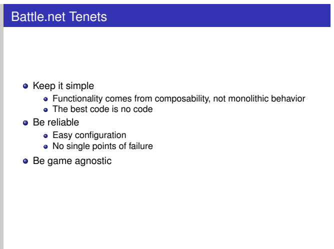
Diablo III Game Distribution by Difficulty

- there’s a bell curve in there
- and some sticky parts
- normal: beginners and farming rift keys
- torment 1: class-specific set items
- torment 6: farming
- sticky parts will change according to design
- load will change over time
- implications for distribution across hardware
- NEXT: Battle.net tenets

2014-06-05

Millions of Games per Hour
└History & Problem Space

└Battle.net Tenets



Battle.net Tenets

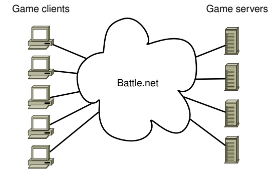
- Keep it simple
  - Functionality comes from composability, not monolithic behavior
  - The best code is no code
- Be reliable
  - Easy configuration
  - No single points of failure
- Be game agnostic

- shipped games are very mediated experiences
- Battle.net back end must be transparent
- simplicity is prerequisite for reliability
- easy operation
- make failures obvious
- don’t do work when things fail
- strenuously avoid game knowledge
- NEXT: system overview (servers are clients)

2014-06-05

Millions of Games per Hour
└─History & Problem Space
└─System Overview

System Overview



- simple diagram but important point: servers are clients
- left: PCs and Macs running on desktops
- right: Linux machines in datacenters
- both are clients
- we trust servers a little more
- Battle.net knows nothing about either side’s operation or semantics
- NEXT: new section - abstracting matching

2014-06-05

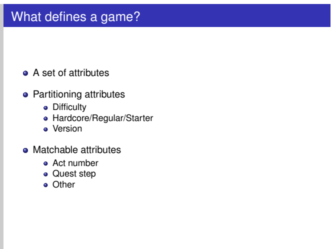
Millions of Games per Hour
└─Abstracting Matching
└─Abstracting Matching

Abstracting Matching

- abstracting matching for multiple games on Battle.net
- NEXT: what defines a game

2014-06-05

Millions of Games per Hour
└─Abstracting Matching
└─What defines a game?



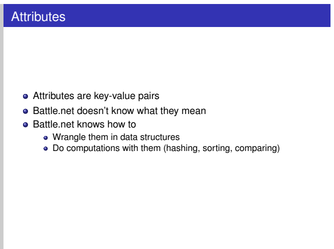
What defines a game?

- A set of attributes
- Partitioning attributes
  - Difficulty
  - Hardcore/Regular/Starter
  - Version
- Matchable attributes
  - Act number
  - Quest step
  - Other

- a set of attributes that define a game (overlay)
- partitioning attributes are (usually) static for a game’s lifetime
- they represent a "hard sharding" of the player base
- they embody game "legality" (overlay)
- matchable attributes are softer, can change during play
- they embody game quality more so than legality
- although we still want high quality games
- also: region/game site
- version useful for dev
- NEXT: attributes

2014-06-05

Millions of Games per Hour
└─Abstracting Matching
└─Attributes



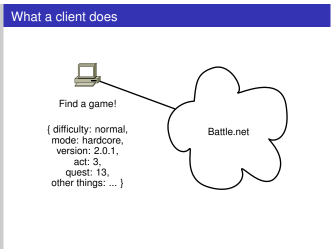
Attributes

- Attributes are key-value pairs
- Battle.net doesn’t know what they mean
- Battle.net knows how to
  - Wrangle them in data structures
  - Do computations with them (hashing, sorting, comparing)

- key is a string
- value is a variant, often a blob of data that is opaque to Battle.net
- we can manipulate them
- we don’t have logic that depends on any particular attribute
- NEXT: what a client does

2014-06-05

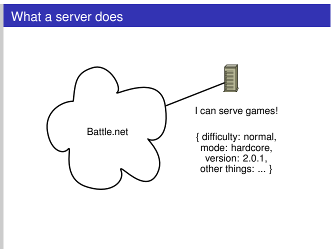
Millions of Games per Hour
└Abstracting Matching
└What a client does


What a client does

- what clients do, conceptually
- the attrs don’t necessarily completely specify a game
- NEXT: what a server does

2014-06-05

Millions of Games per Hour
└Abstracting Matching
└What a server does

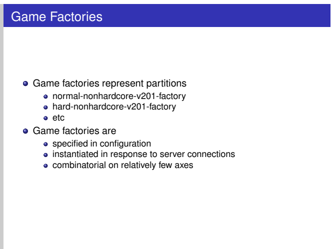

What a server does

- remember a server is also a client of Battle.net
- server attributes are less specific
- eg. in practice perhaps just version
- helpful to have homogeneous server pool to serve all games
- help with resource utilisation when demand is uneven among game types
- NEXT: game factories

2014-06-05

Millions of Games per Hour
└─Abstracting Matching

└─Game Factories



Game Factories

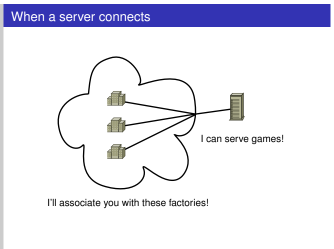
- Game factories represent partitions
  - normal-nonhardcore-v201-factory
  - hard-nonhardcore-v201-factory
  - etc
- Game factories are
  - specified in configuration
  - instantiated in response to server connections
  - combinatorial on relatively few axes

- we want to partition the universe considered for game matching
- naturally we can do this using the partitioning attributes
- leads to the idea of a game factory
- explain what a game factory IS
- we don’t want too many factories
  - segments players unduly
  - combinatorial on attributes
  - tradeoff static vs dynamic
- factories can be instantiated on demand
- NEXT: when a server connects

2014-06-05

Millions of Games per Hour
└─Abstracting Matching

└─When a server connects



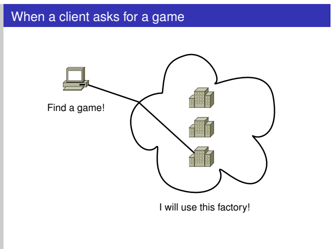
When a server connects

I can serve games!

I’ll associate you with these factories!

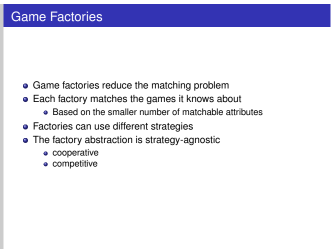
- Battle.net makes factories on demand from config
- add references if they already exist
- for each factory, know which servers can make games for it
- NEXT: when a client finds a game

└When a client asks for a game


When a client asks for a game

- client game choice can be used to select a factory
- cut down the matching space
- NEXT: how factories help

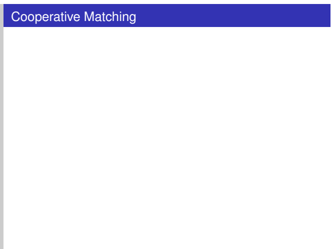
└Game Factories


Game Factories

- Game factories reduce the matching problem
- Each factory matches the games it knows about
  - Based on the smaller number of matchable attributes
- Factories can use different strategies
- The factory abstraction is strategy-agnostic
  - cooperative
  - competitive

- factories reduce the matching problem space by cutting out the static attributes
- nothing about the factory abstraction dictates a matching strategy
- different factories can implement different strats (coop or comp)
- factory may keep track of games running (for coop drop in/out)
- or may not need to (comp join-at-start)
- NEXT: new section - cooperative matching

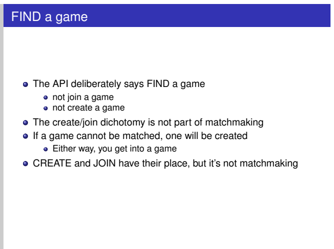
└Cooperative Matching



Cooperative Matching

- in the context of Diablo 3
- NEXT: FIND a game

└FIND a game

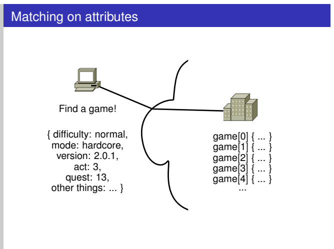


FIND a game

- The API deliberately says FIND a game
  - not join a game
  - not create a game
- The create/join dichotomy is not part of matchmaking
- If a game cannot be matched, one will be created
  - Either way, you get into a game
- CREATE and JOIN have their place, but it’s not matchmaking

- a new way of thinking vs server lists
- min 1 player in game means you can always find a game
- create and join for friend games
- create still needs to go through MM for HW assignment
- coop: a new way of thinking vs skill-based
- matching doesn’t take human time
- no need for waiting/cancellation
- NEXT: the reduced problem

└Matching on attributes



Matching on attributes

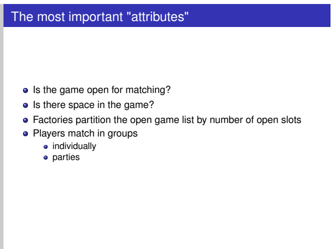
Find a game!

{ difficulty: normal,
mode: hardcore,
version: 2.0.1,
act: 3,
quest: 13,
other things: ... }

game[0] { ... }
game[1] { ... }
game[2] { ... }
game[3] { ... }
game[4] { ... }
...

- the problem reduced, so far
- only dynamic, matchable attributes left
- still may be a lot of games (power law of popularity)
- need to attack the problem further
- NEXT: open/right-size games

└The most important "attributes"

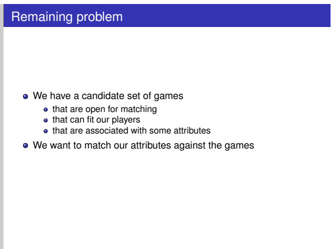


The most important "attributes"

- Is the game open for matching?
- Is there space in the game?
- Factories partition the open game list by number of open slots
- Players match in groups
  - individually
  - parties

- the most important "attributes"
- space and open/closed
- players can find a game individually or in groups
- obvious to keep separate matching pools by number of open slots
- MM knows nothing about party logic
- roles, permissions, etc
- NEXT: what’s left to solve

└Remaining problem

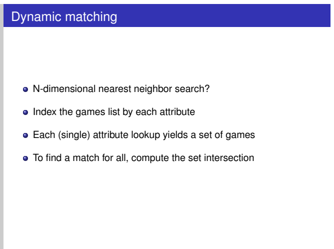


Remaining problem

- We have a candidate set of games
  - that are open for matching
  - that can fit our players
  - that are associated with some attributes
- We want to match our attributes against the games

- factories cut down the space
- cut down the matching universe more by open slots
- this is the remaining problem
- NEXT: dynamic matching

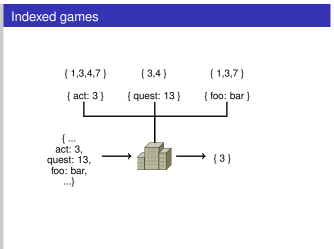
└Dynamic matching



Dynamic matching

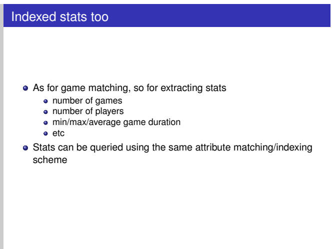
- N-dimensional nearest neighbor search?
- Index the games list by each attribute
- Each (single) attribute lookup yields a set of games
- To find a match for all, compute the set intersection

- at first glance looks like a nearest-neighbour problem
- susceptible to a solution with locality-sensitive hashing?
- but clients don’t have to fully specify games => missing dimensions
- it’s a problem of set building
- take the set of games and index it on each attribute
- NEXT: indexed sets diagram



Indexed games

- each attribute separately indexed and games looked up
- set intersection is the set of games that match the whole query
- NEXT: it works for stats too



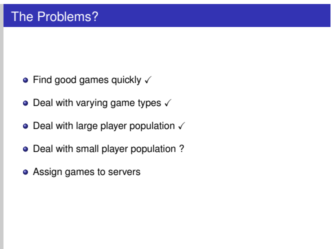
Indexed stats too

- As for game matching, so for extracting stats
  - number of games
  - number of players
  - min/max/average game duration
  - etc
- Stats can be queried using the same attribute matching/indexing scheme

- expose these to the back end
- expose these to players to let them see popularity of their choices
- we can use other logic besides intersection
  - match all (intersection)
  - match any (union)
  - match none (inverse of union)
- useful for stats
- NEXT: what’s solved so far

2014-06-05

Millions of Games per Hour
└Cooperative Matching
└The Problems?



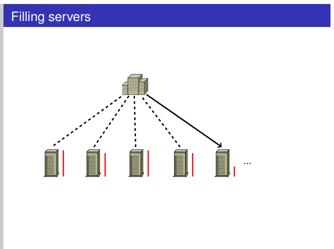
The Problems?

- Find good games quickly ✓
- Deal with varying game types ✓
- Deal with large player population ✓
- Deal with small player population ?
- Assign games to servers

- problems solved so far
- sensible to optimize any MM for scale
- problems at small scale by definition don’t affect many people
- with a small population, game quality may be poor anyway
- NEXT: filling servers

2014-06-05

Millions of Games per Hour
└Cooperative Matching
└Filling servers

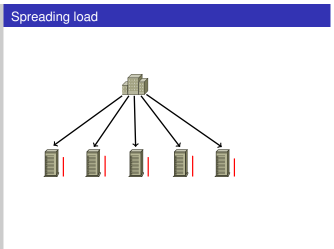


Filling servers

- if architecture allows spinning up and shutting down new servers on demand
- NEXT: spreading load

2014-06-05

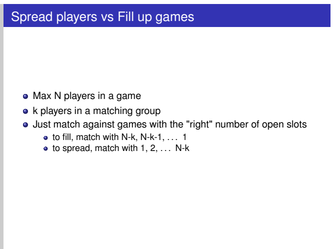
Millions of Games per Hour
└─Cooperative Matching
└─Spreading load


Spreading load

- if architecture has fixed number of servers
- NEXT: spreading/filling games

2014-06-05

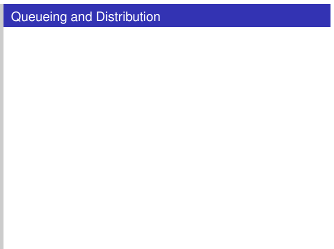
Millions of Games per Hour
└─Cooperative Matching
└─Spread players vs Fill up games


Spread players vs Fill up games

- Max N players in a game
- k players in a matching group
- Just match against games with the "right" number of open slots
  - to fill, match with N-k, N-k-1, . . . 1
  - to spread, match with 1, 2, . . . N-k

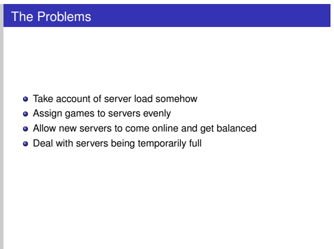
- distinct from server filling choice, game filling choice
- first we did spread the players (for D3)
- most games were empty
- so we switched to filling games, much better
- if you leave a game and rematch, likely to get back in the same game
- NEXT: new section - queueing/distribution

└Queueing and Distribution


Queueing and Distribution

- so we know how to match to make good games
- now we need to think about assigning games to servers
- NEXT: the problems

└The Problems

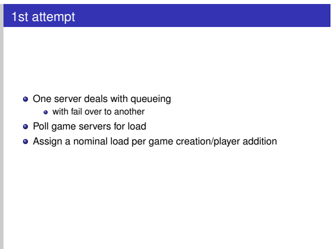

The Problems

- Take account of server load somehow
- Assign games to servers evenly
- Allow new servers to come online and get balanced
- Deal with servers being temporarily full

- why round robin doesn’t work
- bringing servers online dynamically to deal with load
- sometimes game servers crash and come back
- we need some idea of how loaded servers are
- we may need to queue people (fairness)
- NEXT: first attempt

2014-06-05

Millions of Games per Hour
└─Queueing and Distribution
└─1st attempt



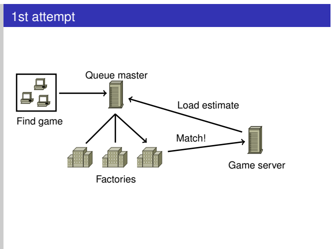
1st attempt

- One server deals with queueing
  - with fail over to another
- Poll game servers for load
- Assign a nominal load per game creation/player addition

- first attempt
- NEXT: 1st attempt diagram

2014-06-05

Millions of Games per Hour
└─Queueing and Distribution
└─1st attempt

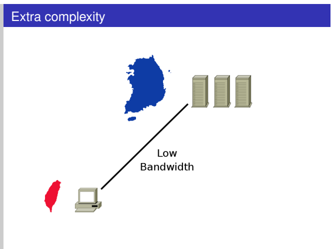


1st attempt

- queueing is separate step from MM
- load from game servers
- let people through as load allows
- NEXT: extra complexity

2014-06-05

Millions of Games per Hour
└Queueing and Distribution
└Extra complexity

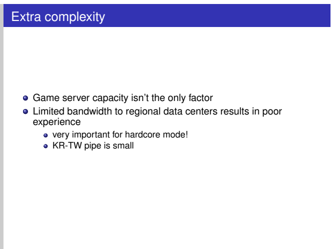


Extra complexity

- complicating factor: bandwidth to regional data centers
- Battle.net works out of US, EU, KR and CN
- some countries have poor ping/BW to regions
- NEXT: extra complexity

2014-06-05

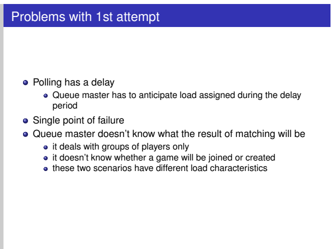
Millions of Games per Hour
└Queueing and Distribution
└Extra complexity



Extra complexity

- Game server capacity isn't the only factor
- Limited bandwidth to regional data centers results in poor experience
  - very important for hardcore mode!
  - KR-TW pipe is small

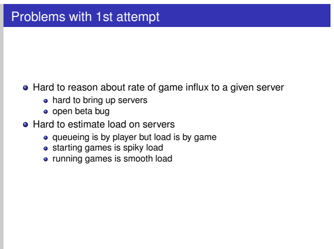
- hardcore mode = permadeath
- protecting game experience
- effectively need two queues (one for game server capacity, one for country capacity)
- complex for mixed groups
- this was a problem even so, we don't really want to gate people's ability to play
- NEXT: problems



Problems with 1st attempt

- Polling has a delay
  - Queue master has to anticipate load assigned during the delay period
- Single point of failure
- Queue master doesn't know what the result of matching will be
  - it deals with groups of players only
  - it doesn't know whether a game will be joined or created
  - these two scenarios have different load characteristics

- polling interval means load is out of date
- queue master has to simulate load according to how it apportions games
- queue master can only estimate load on a factory basis (it doesn't know which server will actually get the game)
- NEXT: more problems



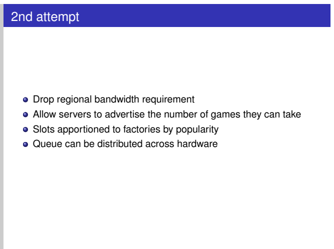
Problems with 1st attempt

- Hard to reason about rate of game influx to a given server
  - hard to bring up servers
  - open beta bug
- Hard to estimate load on servers
  - queueing is by player but load is by game
  - starting games is spiky load
  - running games is smooth load

- queue master doesn't know about MM
- rate of game creation is unknown to it
- queueing is by players (groups) but load is more by game
- game creation or find? unknown to queue master
- game steady state load is a poor model of game creation load
- game creation load is high
- we had to model rate limiting and simulated load - more complexity
- failure case: if server not ready, select another - more complexity
- NEXT: 2nd attempt

2014-06-05

Millions of Games per Hour
└Queueing and Distribution
└2nd attempt



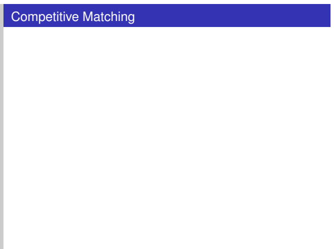
2nd attempt

- Drop regional bandwidth requirement
- Allow servers to advertise the number of games they can take
- Slots apportioned to factories by popularity
- Queue can be distributed across hardware

- simpler!
- no more country queues - we put game servers in those countries (TW, Aus)
- it was hard for game servers to estimate their load
- easier for them to have explicit control over how many games they can afford
- and the rate they can afford
- this is much better, fixes the problem
- NEXT: new section - comp matching

2014-06-05

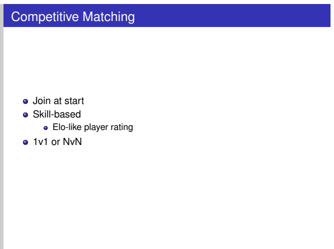
Millions of Games per Hour
└Competitive Matching
└Competitive Matching



Competitive Matching

- NEXT: recap comp match characteristics

└Competitive Matching

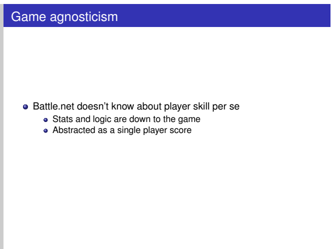


Competitive Matching

- Join at start
- Skill-based
  - Elo-like player rating
- 1v1 or NvN

- competitive matching is very different to cooperative matching
- comp still uses the factory abstraction
- can't join a game midway (design/tech doesn't allow, rating system doesn't allow)
- players expect good matches
- 1v1 is normal happy case
- NvN needs some aggregation of stats to get a good team match
- NEXT: game agnostic

└Game agnosticism



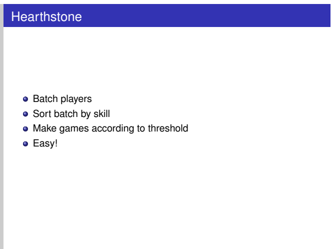
Game agnosticism

- Battle.net doesn't know about player skill per se
  - Stats and logic are down to the game
  - Abstracted as a single player score

- recall Battle.net's core tenets
- protects us from accidentally building specifics that won't work on another game
- NEXT: HS batching

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─Hearthstone

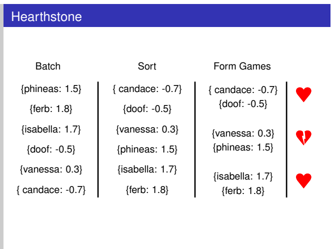

Hearthstone

- Batch players
- Sort batch by skill
- Make games according to threshold
- Easy!

- when all you have is 1v1, competitive matchmaking is comparatively easy
- this is what HS does at a basic level
- NEXT: diagram

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─Hearthstone


Hearthstone

Batch	Sort	Form Games	

{phineas: 1.5}	{ candace: -0.7}	{ candace: -0.7}
		{doof: -0.5}
			♥

{ferb: 1.8}	{doof: -0.5}		

{isabella: 1.7}	{vanessa: 0.3}	{vanessa: 0.3}
		{phineas: 1.5}
			💔

{doof: -0.5}	{phineas: 1.5}		

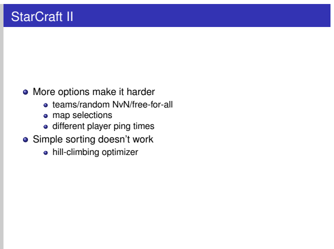
{vanessa: 0.3}	{isabella: 1.7}	{isabella: 1.7}
		{ferb: 1.8}
			♥

{ candace: -0.7}	{ferb: 1.8}		

- leftover players are thrown back into the next batch and have their threshold relaxed
- HS is not latency-sensitive (turn-based)
- NEXT: SC2 is harder

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─StarCraft II



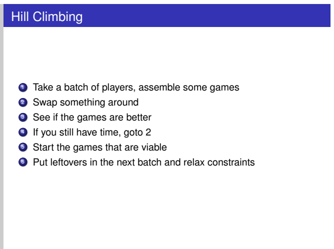
StarCraft II

- More options make it harder
  - teams/random NvN/free-for-all
  - map selections
  - different player ping times
- Simple sorting doesn’t work
  - hill-climbing optimizer

- SC2 is a different animal
- With more options, simply sorting by skills doesn’t work
- In particular, dealing with teams requires some thought
- there are always more maps possible than can be vetoed
- NEXT: hill-climbing algo

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─Hill Climbing

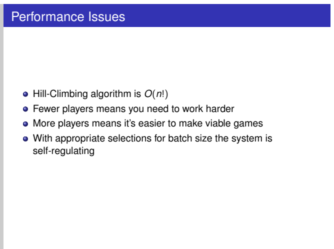


Hill Climbing

1. Take a batch of players, assemble some games
2. Swap something around
3. See if the games are better
4. If you still have time, goto 2
5. Start the games that are viable
6. Put leftovers in the next batch and relax constraints

- basic hill-climbing optimization algorithm
- make a batch based on size or after a time has elapsed
- NEXT: hill-climbing perf

└Performance Issues

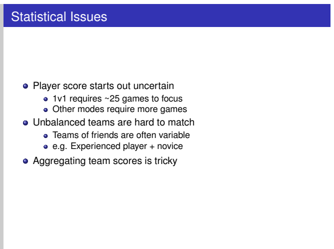


Performance Issues

- Hill-Climbing algorithm is $O(n!)$
- Fewer players means you need to work harder
- More players means it's easier to make viable games
- With appropriate selections for batch size the system is self-regulating

- number of permutations is O(n!) wrt batch size
- tradeoff batch size and quality of match
- you can thread batches
- presort players before batching and threading
- fewer players -> have to do more work (higher variance)
- more players -> have to do less work
- NEXT: stats issues

└Statistical Issues

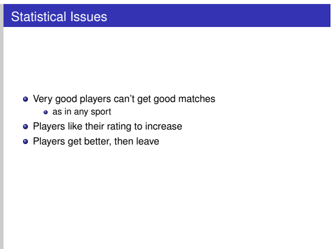


Statistical Issues

- Player score starts out uncertain
  - 1v1 requires ~25 games to focus
  - Other modes require more games
- Unbalanced teams are hard to match
  - Teams of friends are often variable
  - e.g. Experienced player + novice
- Aggregating team scores is tricky

- players take time to home in on true rating
- 1v1 homes in quickest, other types may be quite slow
- rating systems tend to be based on 1v1 games (chess) with binary outcomes
- bell curve of players, variation between players gives likelihood of outcome
- team skill is non-linear: more or less than sum of parts
- NEXT: more stats issues

└Statistical Issues

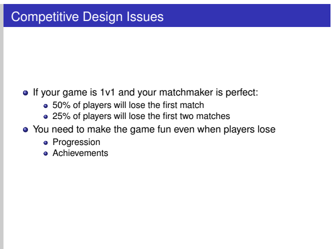


Statistical Issues

- Very good players can't get good matches
  - as in any sport
- Players like their rating to increase
- Players get better, then leave

- a good match is within 0.2 sigma
- far end of bell curve don't have many people to match against
- practice partners and tournament play
- trade off time to get match vs quality of match
- hide the true rating
- tiers give progression
- seasons give a reset
- rating points drain
- NEXT: design issues

└Competitive Design Issues



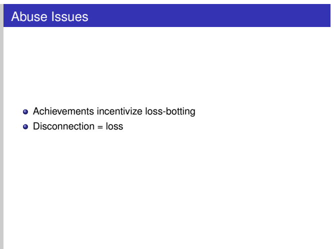
Competitive Design Issues

- If your game is 1v1 and your matchmaker is perfect:
  - 50% of players will lose the first match
  - 25% of players will lose the first two matches
- You need to make the game fun even when players lose
  - Progression
  - Achievements

- design of the game affects competition subjectively
- HS does a good job of this
- NEXT: abuses

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─Abuse Issues



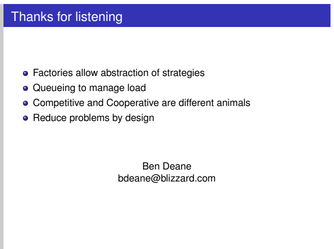
Abuse Issues

- Achievements incentivize loss-botting
- Disconnection = loss

- loss-botting: bots quit early, and pick up wins
- loss bots drop to same rating then trade wins
- if players are at that rating, they play against a lot of bots
- require min game time
- abuse story: crash clients, patch own, win games in 5s
- crashes, game length oddities, statistical oddities
- NEXT: final slide

2014-06-05

Millions of Games per Hour
└─Competitive Matching
└─Thanks for listening



Thanks for listening

- Factories allow abstraction of strategies
- Queueing to manage load
- Competitive and Cooperative are different animals
- Reduce problems by design

Ben Deane
bdeane@blizzard.com

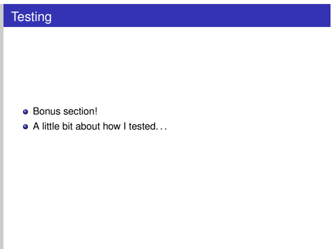
- remember these points:
- abstraction
- queueing/distribution choices
- comp requires heavy statistics
- design can reduce the problem space massively
- thanks for listening
- NEXT: bonus section

2014-06-05

Millions of Games per Hour

└─Competitive Matching

└─Testing



Testing

- Bonus section!
- A little bit about how I tested. . .

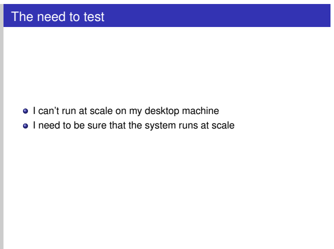
- NEXT: testing needs

2014-06-05

Millions of Games per Hour

└─Competitive Matching

└─The need to test



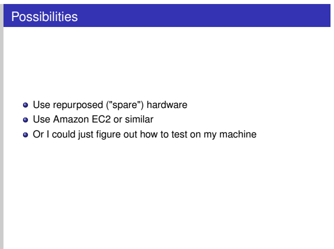
The need to test

- I can't run at scale on my desktop machine
- I need to be sure that the system runs at scale

- I needed to be sure that the system would work as planned at scale
- NEXT: testing choices

2014-06-05

Millions of Games per Hour
└Competitive Matching
└Possibilities



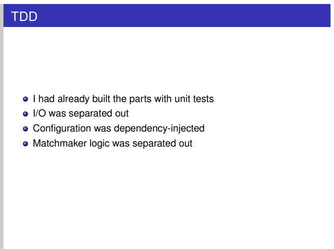
Possibilities

- Use repurposed ("spare") hardware
- Use Amazon EC2 or similar
- Or I could just figure out how to test on my machine

- using DR hardware is viable, we do that
- it takes a lot to set up and run
- it is important for full system/integration testing
- we don’t like to run code outside of the building so EC2 was out
- NEXT: TDD

2014-06-05

Millions of Games per Hour
└Competitive Matching
└TDD



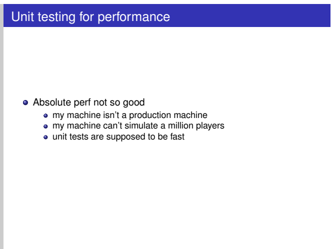
TDD

- I had already built the parts with unit tests
- I/O was separated out
- Configuration was dependency-injected
- Matchmaker logic was separated out

- I used TDD
- everything was testable already
- I just needed to rig perf tests
- NEXT: absolute perf testing

2014-06-05

Millions of Games per Hour
└Competitive Matching
└Unit testing for performance



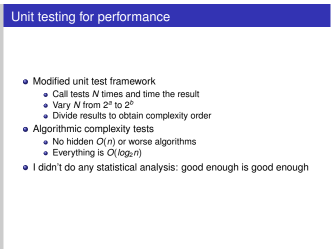
Unit testing for performance

- Absolute perf not so good
  - my machine isn't a production machine
  - my machine can't simulate a million players
  - unit tests are supposed to be fast

- absolute performance doesn't tell me much
- my desktop machine isn't real
- different HW, different OS
- I still can't simulate scale quickly
- I need algorithmic complexity guarantees
- NEXT: algo testing

2014-06-05

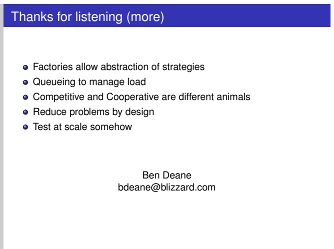
Millions of Games per Hour
└Competitive Matching
└Unit testing for performance



Unit testing for performance

- Modified unit test framework
  - Call tests *N* times and time the result
  - Vary *N* from 2^a to 2^b
  - Divide results to obtain complexity order
- Algorithmic complexity tests
  - No hidden $O(n)$ or worse algorithms
  - Everything is $O(log_2 n)$
- I didn't do any statistical analysis: good enough is good enough

- vary input size, compare run times
- bucket the times to O(1), O(log n), O(n) etc
- very easy to accidentally introduce an O(n) library call
- strictly this is not really a unit test - it's not guaranteed to work every time
- but good enough is good
- NEXT: final slide (really)



Thanks for listening (more)

- Factories allow abstraction of strategies
- Queueing to manage load
- Competitive and Cooperative are different animals
- Reduce problems by design
- Test at scale somehow

Ben Deane
bdeane@blizzard.com

- remember these points
- thanks for listening