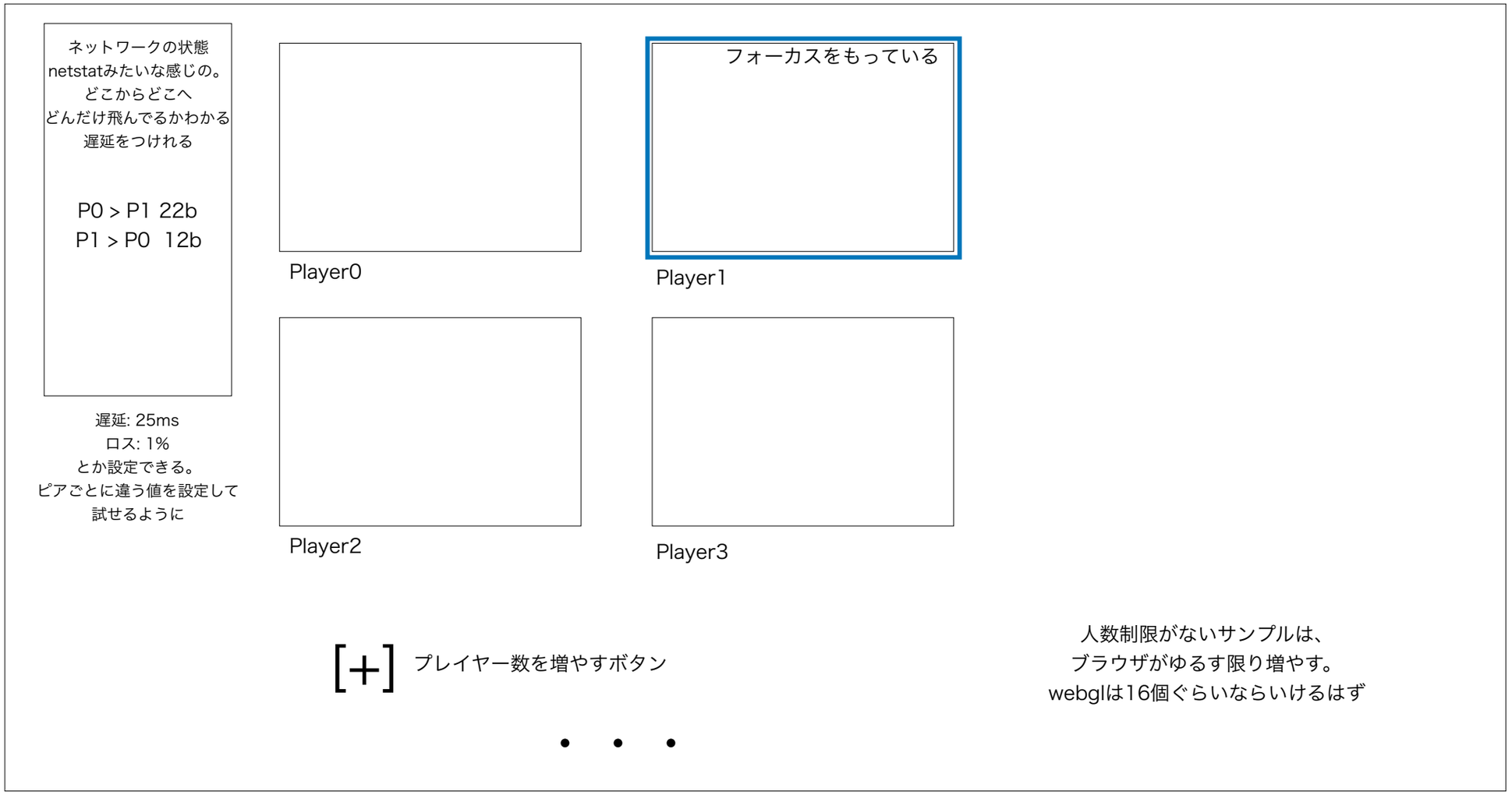
サンプルゲームの構成案

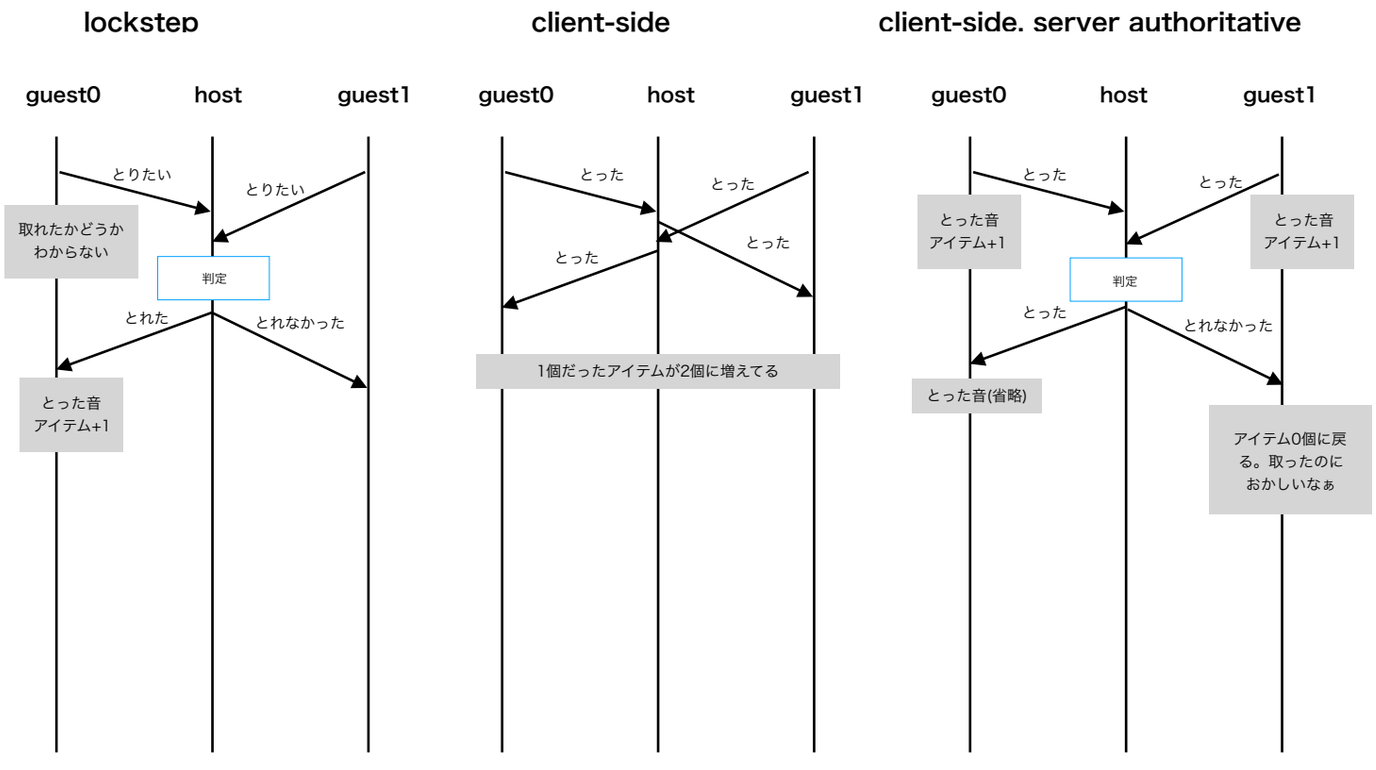
・ブラウザ用 (WebGL)
・実際に通信をしないが、仮想ネットワークを使ってブラウザ内で通信の遅れを試せる。通信の形式は一般化できないので。
・すべてホスト/ゲスト型トポロジーで実装する(ネットワーク構造とは独立)
・人数分のcanvasを用意して、1画面で全部見れる。
・ゲーム内容は単純化するが、オブジェクトの物量はできるだけ近づける。
・操作は,P0がWASD+ZX+クリック P1が矢印+P/;(クリック) P2以降は操作なし



同期モードの選択

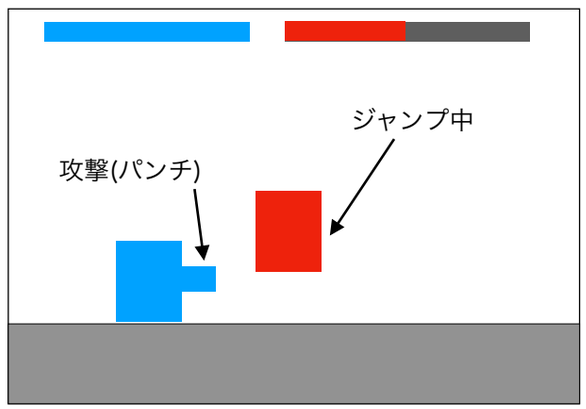
1. lockstep(vol1では同期式)
   全員の入力をサーバに揃えてから処理し、結果を全員に配布する。最もラグの大きいピアがボトルネックになる。
2. guest-side(vol1では非同期式)(client-side)
   常にクライアントつまりゲスト側で判定してそれを結果として採用する。
   FPSでは、撃った側で当たった判定もするので、ラグがある場合、撃たれた人は、隠れた後に撃たれてしまう。
3. guest-side + host-authoritative
   ゲスト判定後ホスト判定で上書きする。
   撃った側で当たった判定もするが、ホスト(サーバ)にも撃ったことを送る。ホストには撃たれた側の隠れる動作が先に届くので、撃たれない結果が最終結果になる。　撃った側では、撃ったはずがダメージが入らないように見える。撃たれた側は、撃たれてないように見える。
4. GGPO
   決定論的ゲームにおいて、client-side, server authoritativeにさらに予測入力+ゲーム全体のロールバック+再計算の機構をつけたもの

ある1個のアイテムをguest0, guest1が同時に取った場合



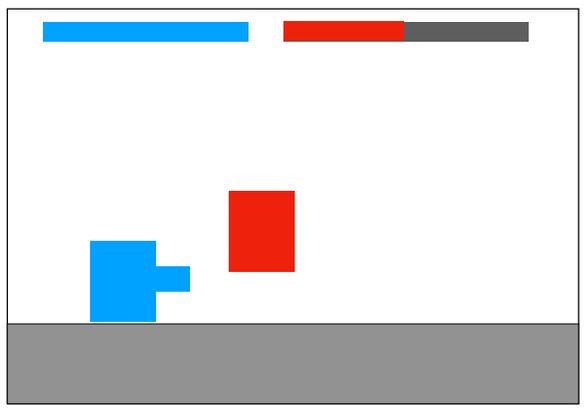
サンプルゲームの構成案

1vs1 格闘ゲーム (GGPO-like)



Player0 (host)

移動:ADジャンプ:W 攻撃S



Player1

移動:←→ジャンプ:↑ 攻撃:↓

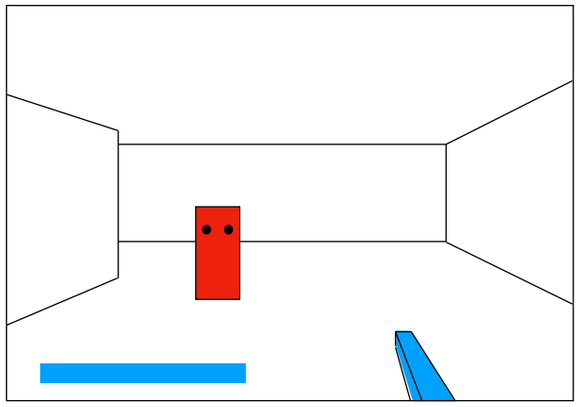
キーボードを使って2人までは同時にプレイできる。

1. 純粋なlockstep
2. GGPO
3. client side
4. client side+server authoritative

この4つで切り替えて試せたら理想的

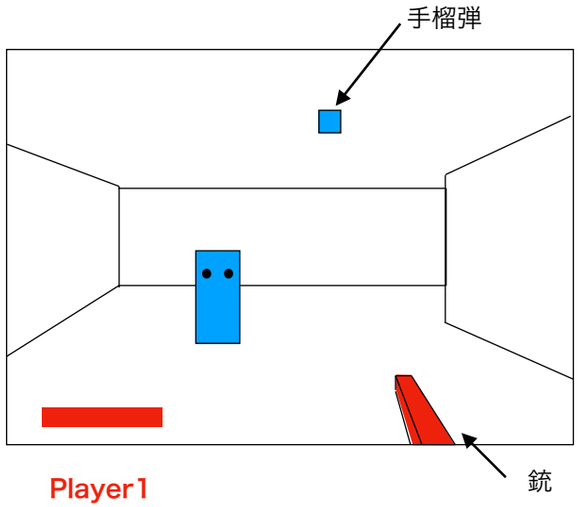
サンプルゲームの構成案

FPS：狭いフィールドでの少人数・高速な撃ち合い



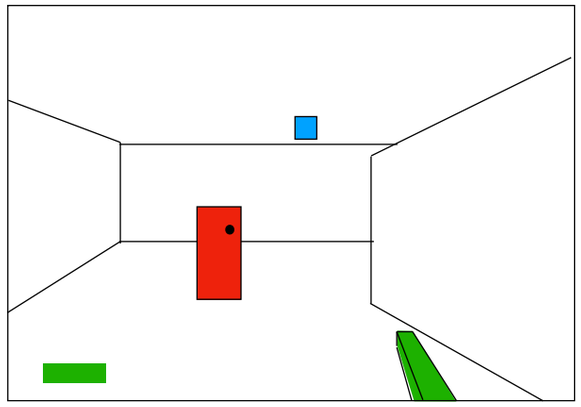
Player0 (host)

移動: WASD 射撃:z 手榴弾:x



Player1

移動: ↑↓→←D 射撃:o 手榴弾:p



Player2

操作なし or ランダム

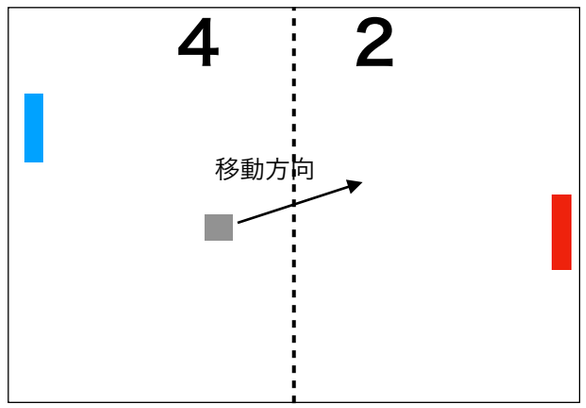
1. lockstep
2. client-side
3. client-side, server authoritative

を切り替えて試せる。

射撃は高弾速、手投げ弾は遅い
物理シムは各ピアで行う。
すべてbcast

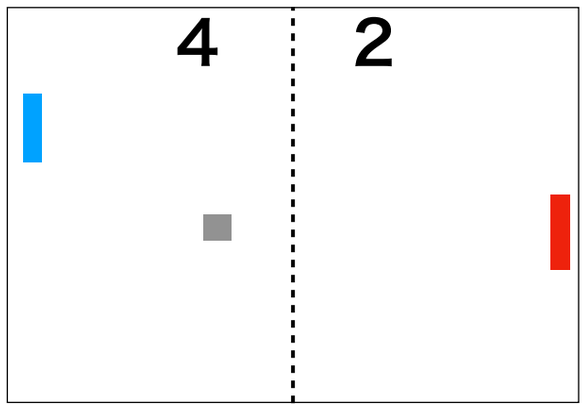
サンプルゲームの構成案

1vs1 PONG 的な打ち返しゲーム



Player0 (host)

移動:WSジャンプ:W 攻撃S



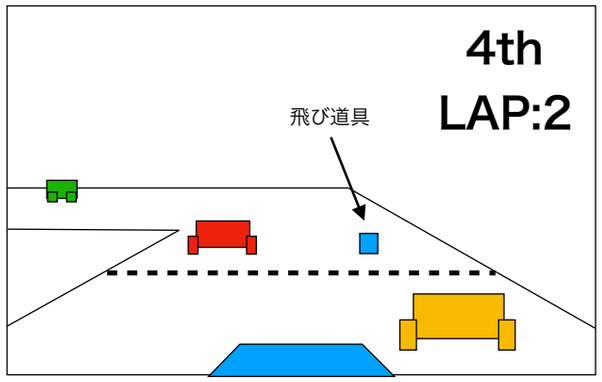
Player1

移動:↑ ↓

1. lockstep
2. GGPO
3. 非決定論的,clientside
4. 非決定論的,clientside+server authoritative

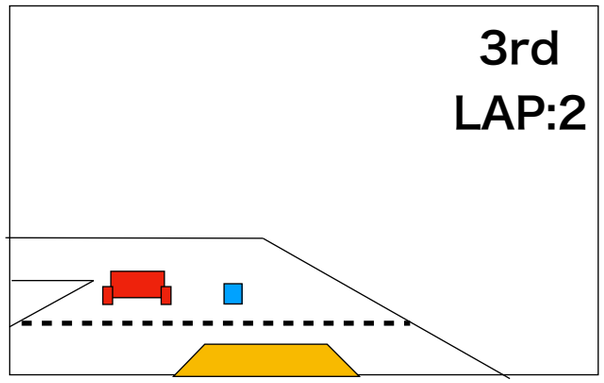
サンプルゲームの構成案

マリカー的な　飛び道具武器ありドライビングゲーム



Player0 (host)

移動: WASD 攻撃:z



Player1

移動: 矢印 攻撃:p

Player3

操作なし or ランダム

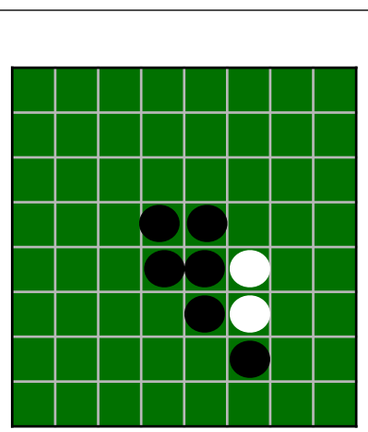
1. lockstep
2. client-side : desyncあり、車同士の衝突を含めてローカルピアですべて判定して結果だけbcast。自分が撃った飛び道具は自分で判定、結果をほかにおくる
3. clientside+server auth

ラグ動画

https://www.youtube.com/watch?v=526ENy4lEqw

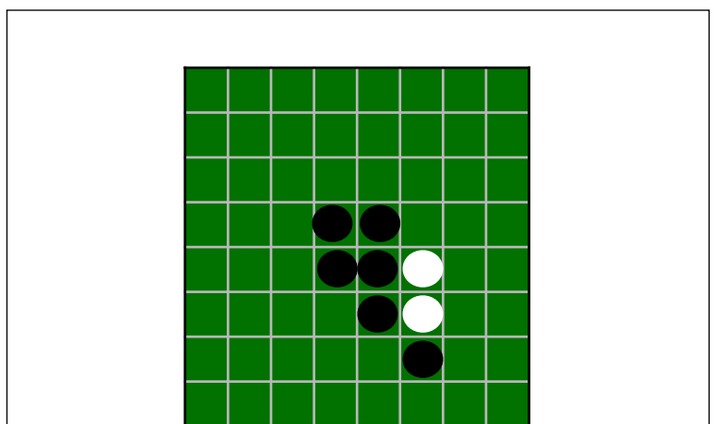
mario cart 8 でラグからのdesyncによって、当たってないのに当てられたのがよくわかる動画。
自分が当てた感覚を、当てられてないことよりも優先していることがわかる。

ターンベースボードゲーム　リバーシ (2P)



Player0 (host)

クリックで置く

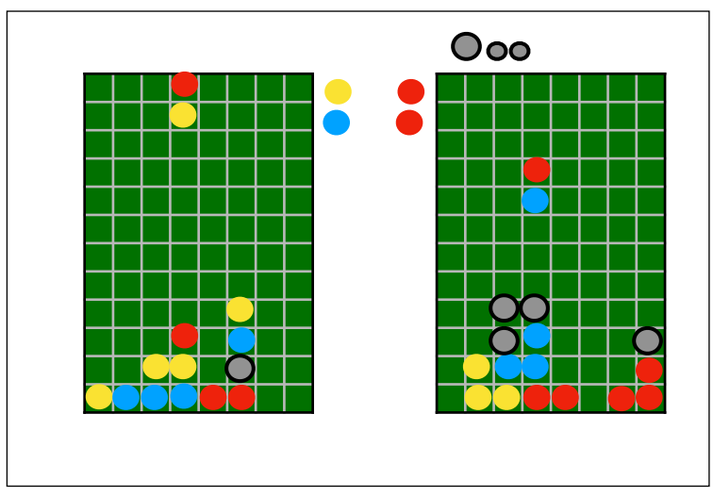


Player1

クリックで置く

1. lockstepのみ．数十ミリ秒の遅れが問題になることはないので、完全にクライアント・サーバモデルで、
ゲストはホストの状態のviewerとして実装で問題がない。

リアルタイム落ちものの対戦 (GGPO-like)



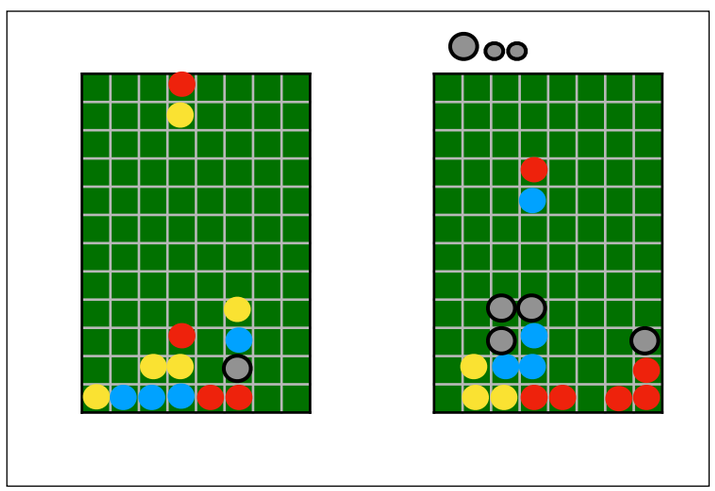
Player0 (host)

左側を操作 移動: ASD 回転:w

1. lockstep
2. GGPO
3. client-side
4. client-side+server autho

ぷよぷよeスポーツでは、新しいぷよの出現タイミングと、
消したタイミング(おじゃまぷよを送るタイミング)について、
ロールバック付き同期をしているようだ。
そのタイミングでラグが発生していると進行が止まる。
それ以外のタイミングではdesyncありでなめらかに進行している。

相手の動きは同期する必要はないが、対戦のために必要な情報なので、
純粋に演出として、適当なFPSで同期・描画すればよい。
ぷよぷよeスポーツなら20FPSぐらい



Player1

右側を操作　移動:←↓→　回転:↑

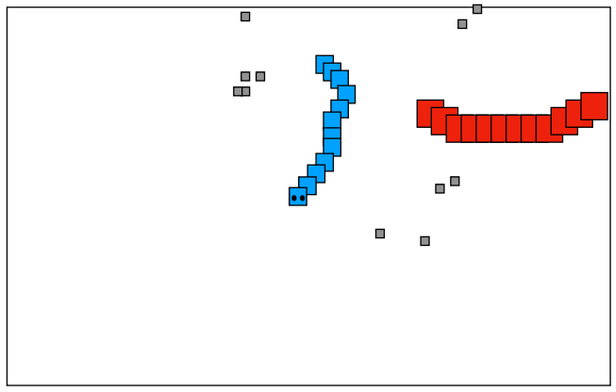
ラグ動画

https://www.youtube.com/watch?v=yBuTunu-pG8

12:10あたりからのゲーム開始直後のひどいラグが参考になる。
ぷよ消しと出現のときに止まることがわかる。
止まってるときは相手側の演出もとまっている。

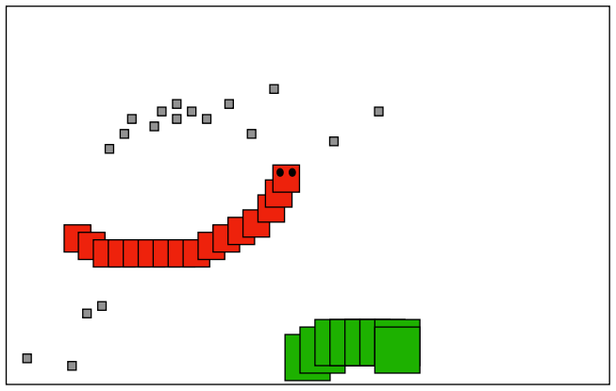
サンプルゲームの構成案

エサを取って太るタイプのioゲーム (不完全同期)



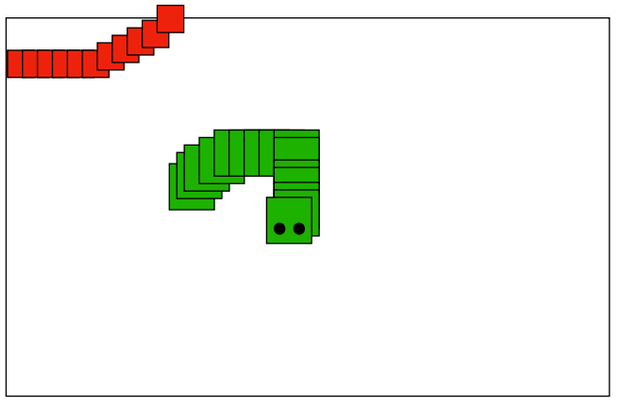
Player0 (host)

移動: AD



Player1

移動:左右



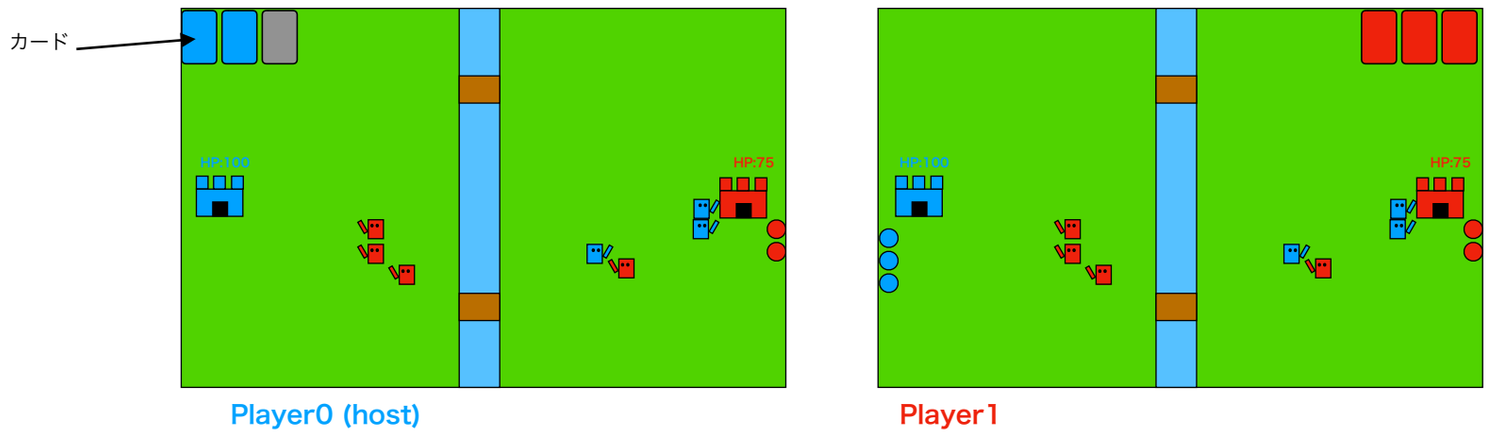
Player2

操作なし or ランダム

1. client side

ほかのもっと厳密な例は、いらんだろ。。というかんじ

1vs1で兵隊を送り合う、遅いペースのリアルタイムカードPvPゲーム （クラロワ)



ユニット設置: クリック

ユニット設置: クリック

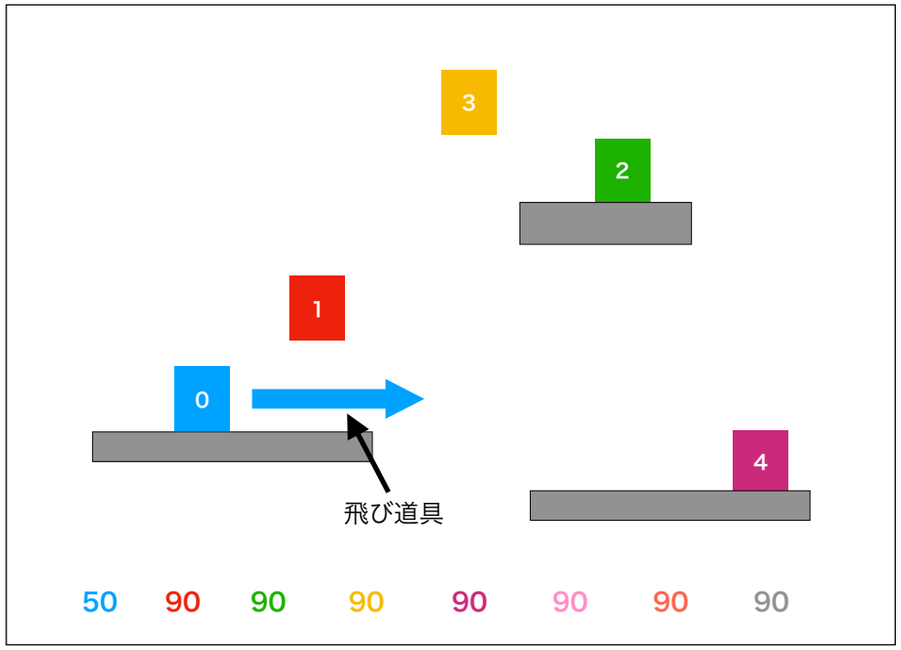
1. lockstep
2. client side
3. client side+sv autho

クラロワは通信ケーブルを一瞬抜くと全体が停止するので、lockstepだとわかる。
ただし抜いた瞬間から少しは操作できるのでlockstepだけでなく、ローカルsimをやっていることがわかる。
ラグから回復したあと、自分側が盛大に巻き戻るときと、そうでないときがある。
そのことから、基本はlockstep + ホストによる上書きとわかる。
通常状態のラグ(~200ms)は、ユニットを配置したときのディレイ(時計)表示で吸収してるはず。

https://www.youtube.com/watch?v=f7MI8Ga2C-0

ラグ動画 14秒ぐらいのところで、ローカルシムの結果(砦壊れず)をホストが上書き(壊れ)　が起きている。
これは典型的なゲスト側のラグ。

スマブラ的な 3人以上用小マップ格闘ゲーム

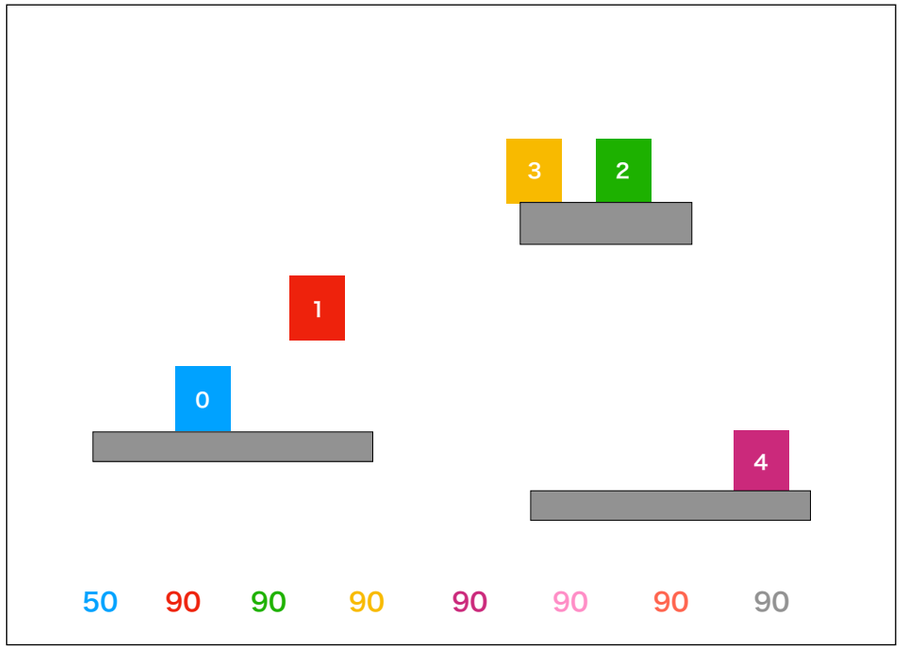


Player0 (host)

移動:ADジャンプ:W 攻撃S 飛び道具z

攻撃OR飛び道具でダメージを与える。

1. lockstep
2. client side
3. client side+sv autho



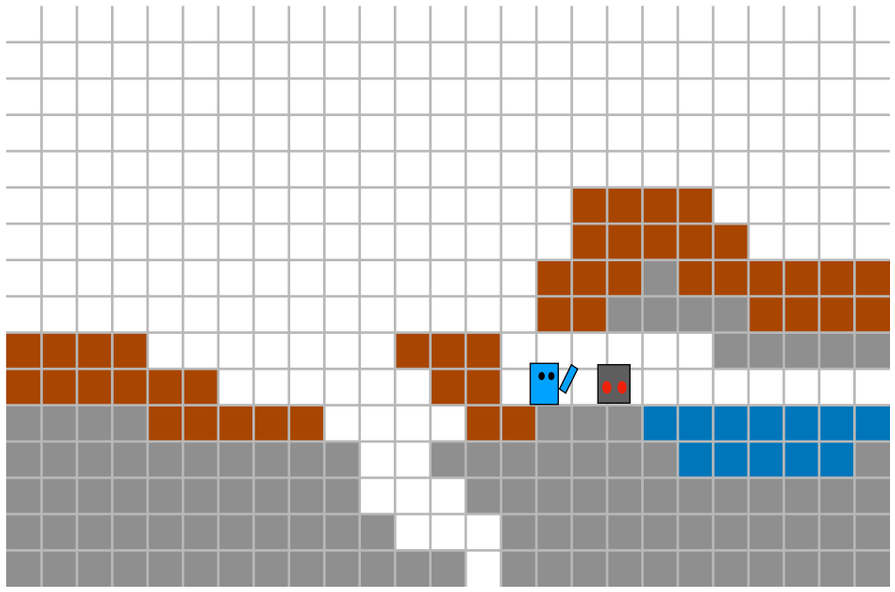
Player1

移動:矢印 攻撃↓　飛び道具p

Player2　・・・

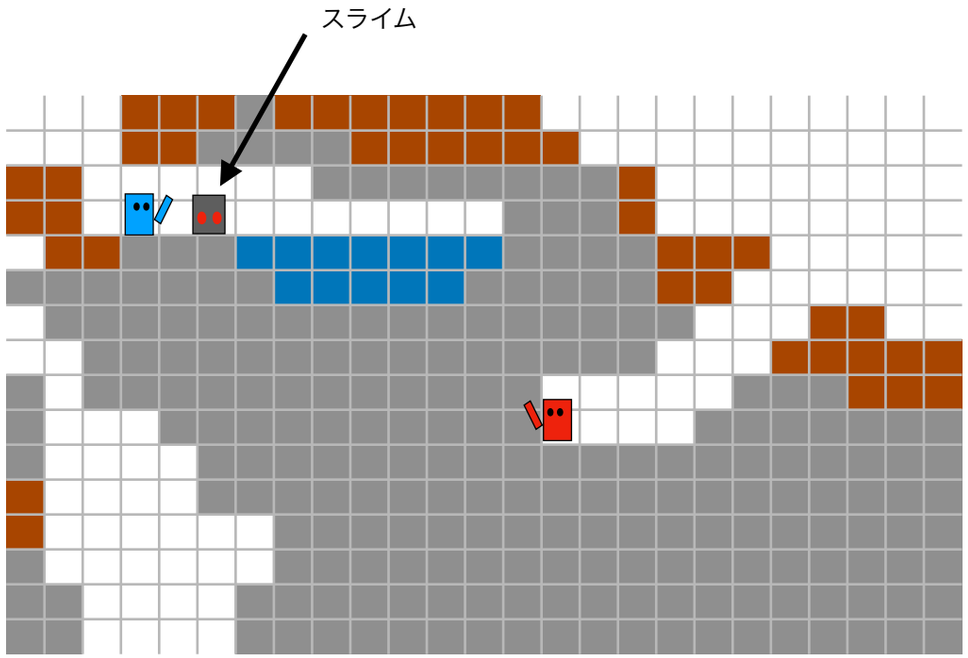
サンプルゲームの構成案

テラリアみたいな2Dサンドボックス　N人用



Player0 (host)

移動: WASD 攻撃/掘る:z



Player1

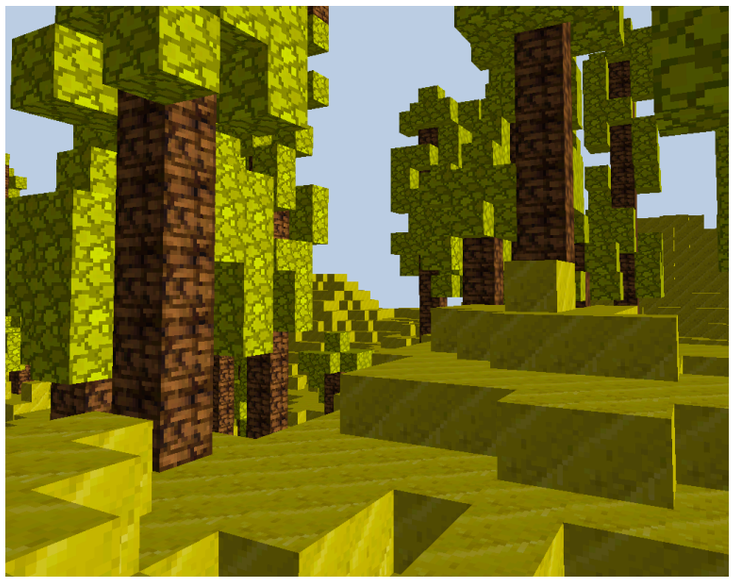
移動:矢印 攻撃/掘る:p

モブは大量にならない。ホストがスポーンさせる。ピアがSimするが、
ホストからの結果で上書き。攻撃判定もすべてホストでやる

1. client side
2. client side+sv autho

サンプルゲームの構成案

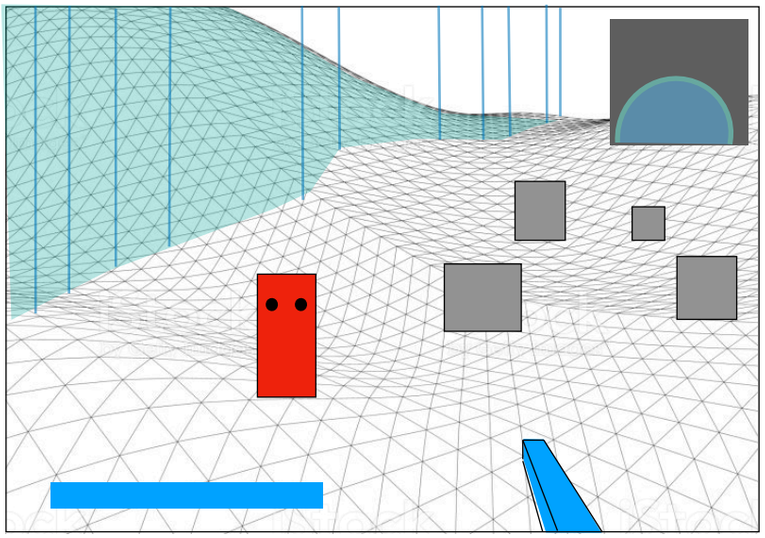
マイクラみたいな3Dサンドボックス　N人用



2Dのが3Dになってるだけ。2Dだけでいいかも

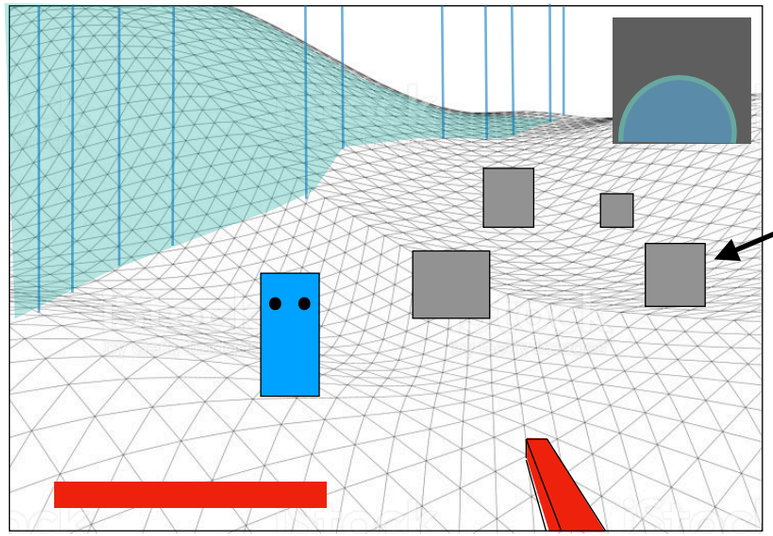
1. client side
2. client side+sv autho

多人数バトロワゲーム
広大マップ、大量の破壊可能コリジョンありオブジェクト、
安全範囲が縮小、銃での撃ち合いPvP



Player0 (host)

移動: WASD 撃つ:z 置く:x



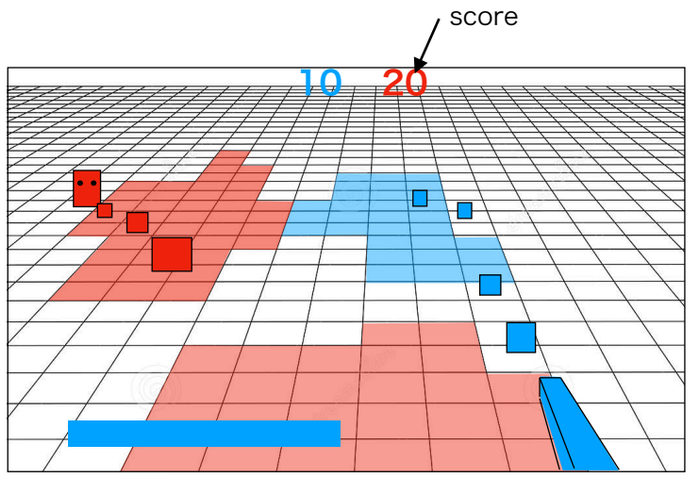
Player1

移動: 矢印 撃つ:p 置く: ;

MMOGサンプルとの違いは、
物理の初期化以外の動き全部と、衝突の判定の全部をクライアントで行う

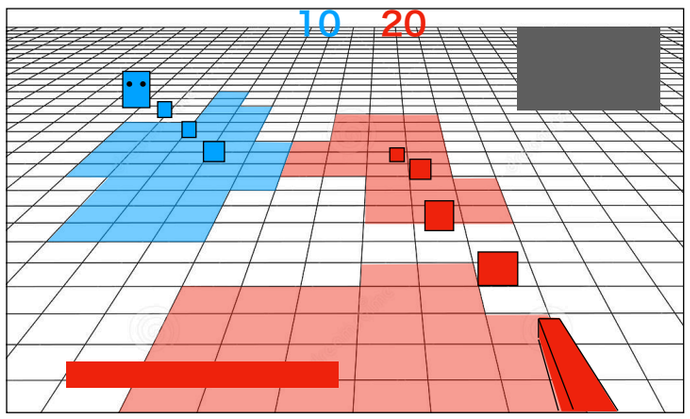
1. client side
2. client side+sv autho

フィールドを撃って塗るゲーム



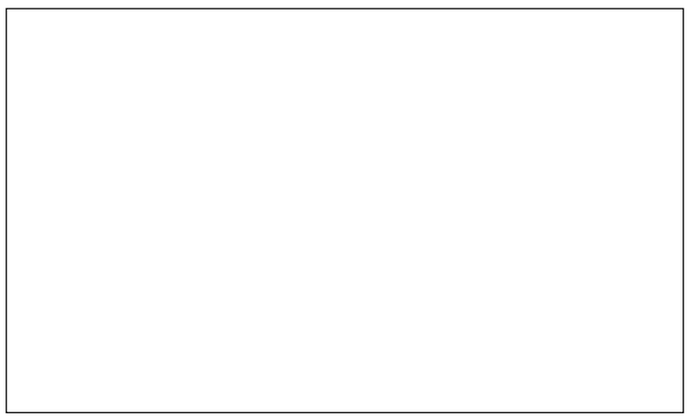
Player0 (host)

移動: WASD 撃つ:z



Player1

移動: 矢印 撃つ:p



Player2 以降　(赤青選択) 操作なし

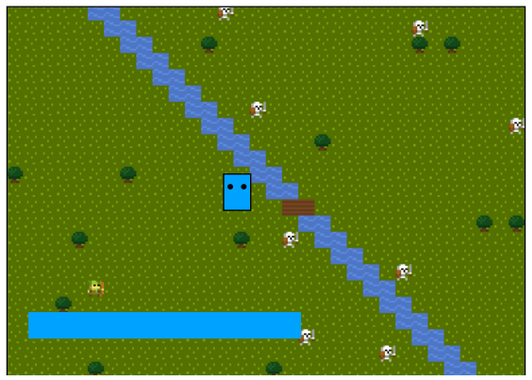
地面を撃つと塗り、敵チームを撃つとダメージ、
HP0で強制リスポーン

塗りは各ピアで塗ってからホストに送信して最終判定、ホストは結果をbcast、ゲストは常に受け入れる。
ホストでのPCの位置からあまりに離れると戻される。
人数が多いのでlockstepはしない。desyncしっぱなし+ホストによる強制上書き。

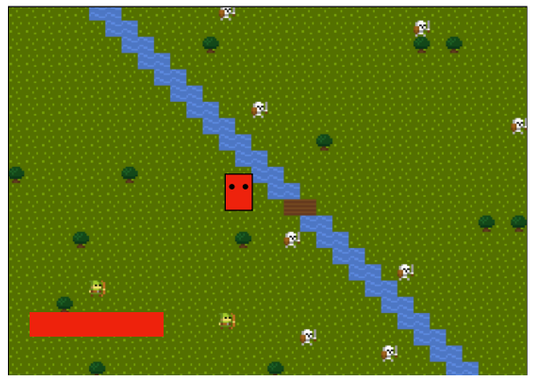
1. lockstep
2. client side
3. client side+sv autho

MMO(RP)G (master server / client viewer)

サーバー
mobをpopして動かす。 全判定をおこなう
512x512程度の広大なマップで
mobとPCの位置を見れる
ダミーのPCを100ぐらい動かしておくか



Player0
移動: wasd 攻撃:z
クライアント側で先行してsimし、
サーバーの処理と不一致だったら戻す(overwrite)



Player1

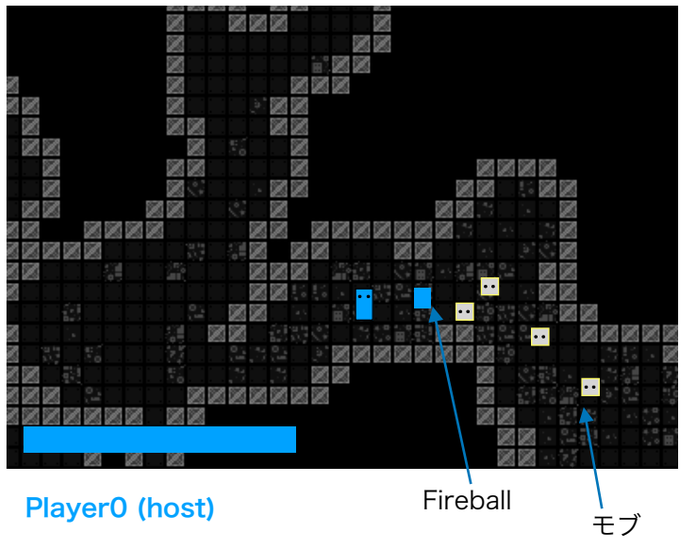
移動: 矢印 攻撃:p

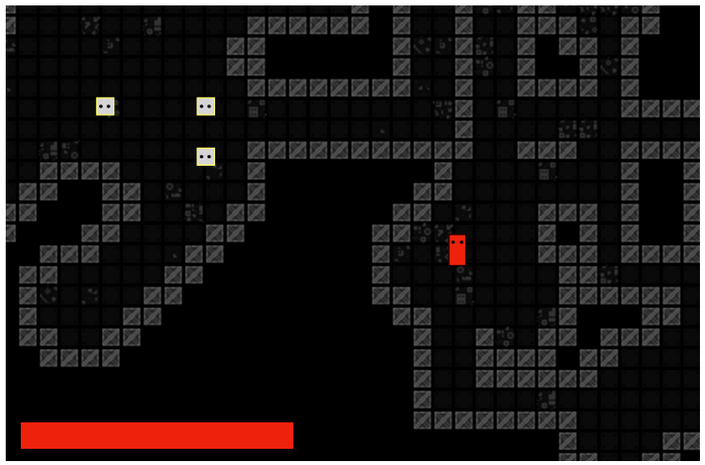
PUBGとの違いは、完全に全部をサーバで行う。
その分物理挙動はシンプルにしておく(実際必要だから)

1. lockstep

ディアブロ的ハクスラ (MOサンプルの移植)

PUBGと近いが、モブが多く、モブの動き(物理シム)と判定をどうするかのサンプルが必要



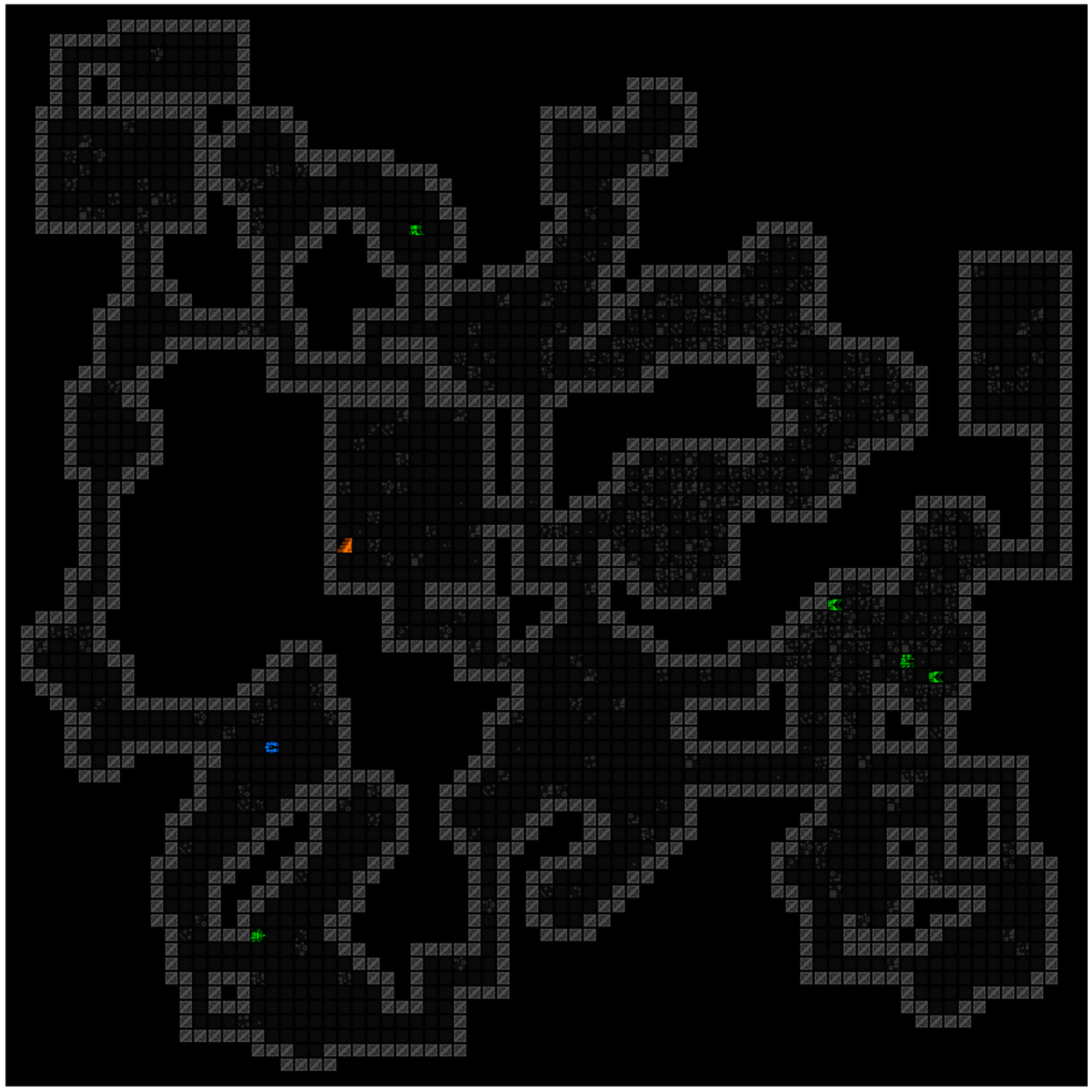


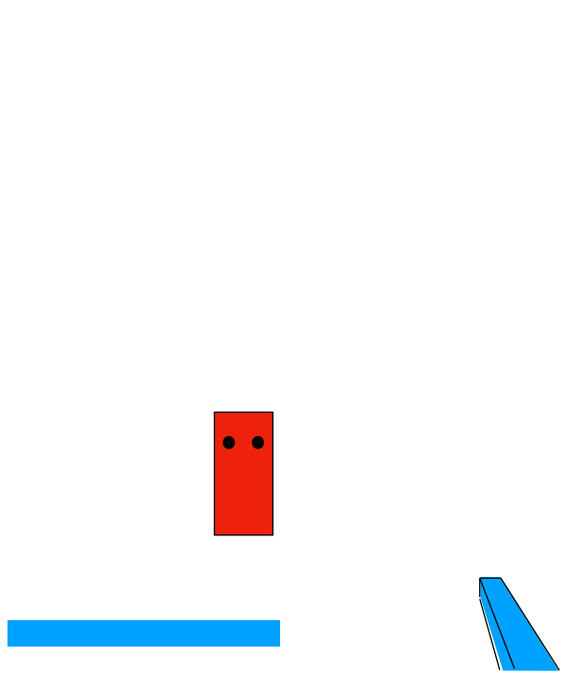
移動: WASD 殴る:z 矢:クリック

移動: 矢印 殴る:p 矢: クリック

PUBGと近いが、モブが多く(合計で300以上)、モブの動き(物理シム)と判定をどうするかのサンプルが必要.
モブのAIは各ピアで行う。判定は各ピアで自己中心的に行う。
挙動中のモブの数を少なく保つため、一定距離離れたモブは眠らせる。
敵を倒すとコインが出る。コイン取得の判定はホスト判定からのrollback

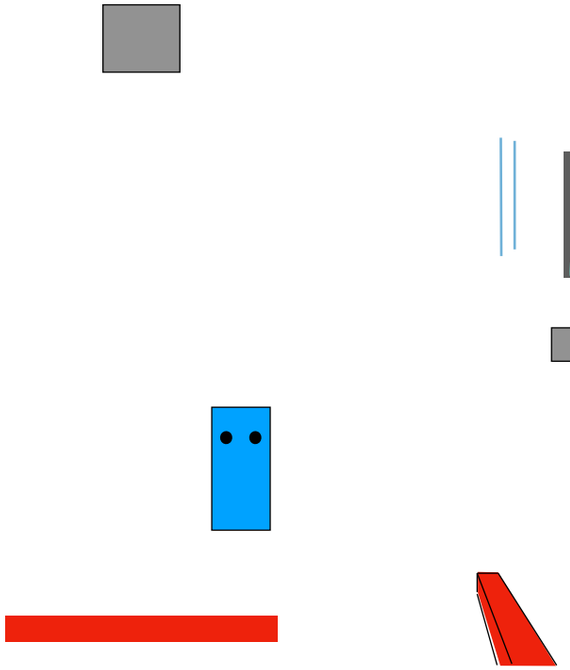
1. lockstep
2. client side
3. client side+sv autho





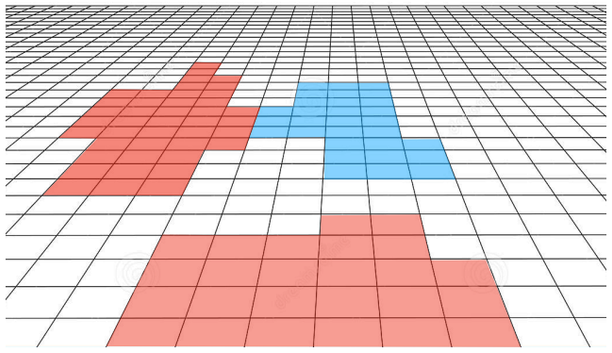
Player0 (host)

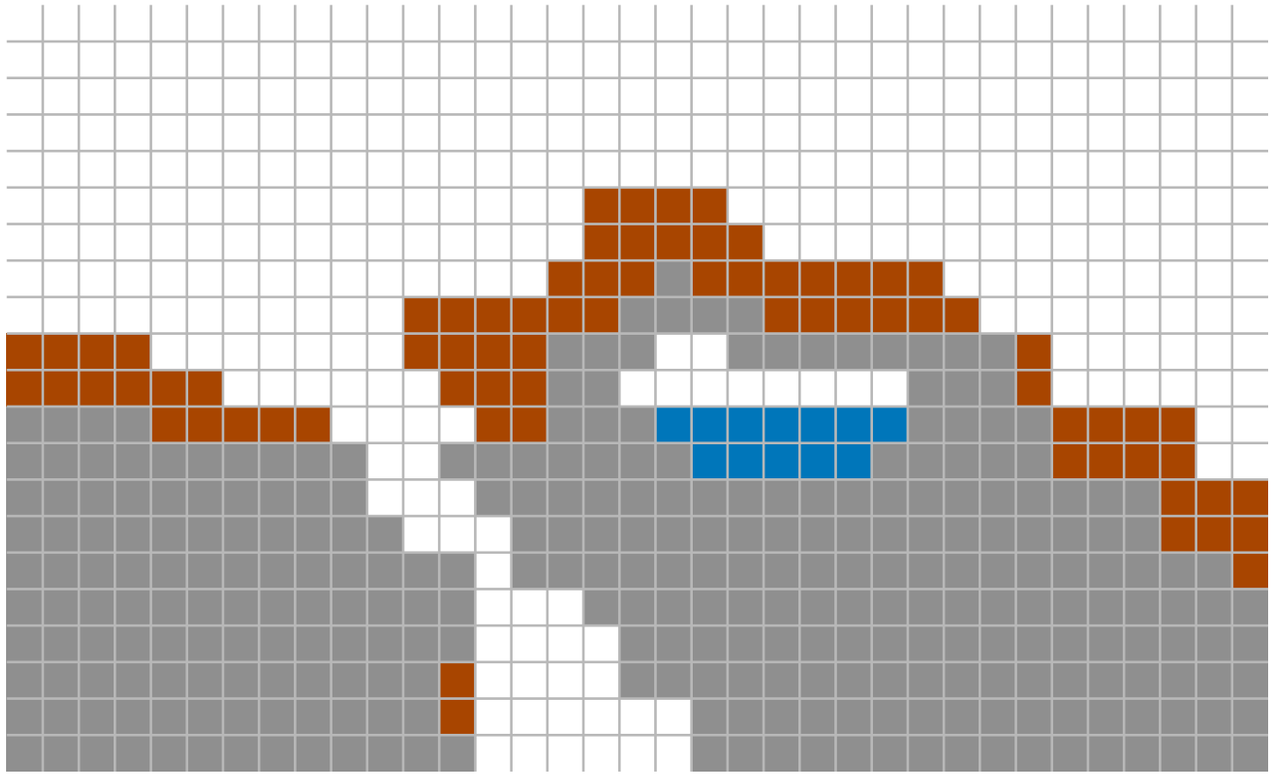
移動: WASD 撃つ:z 置く:x

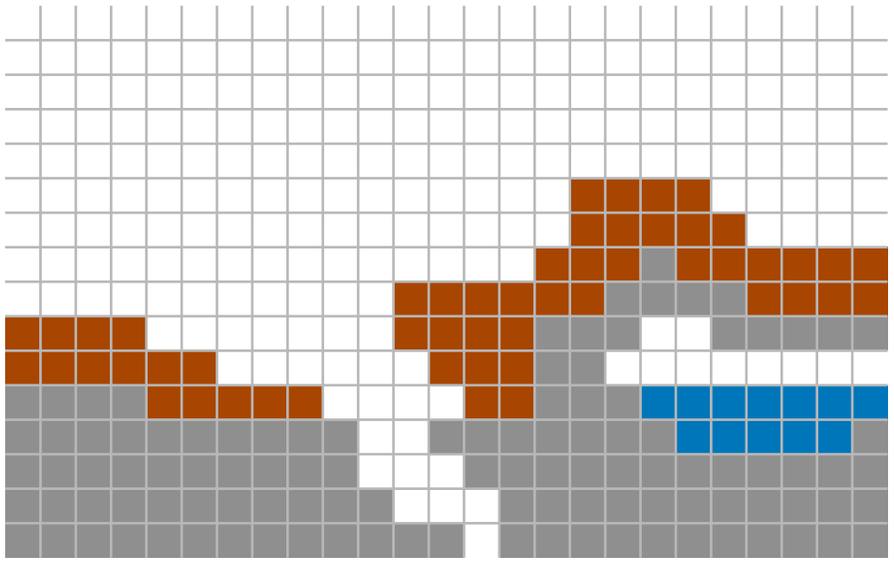


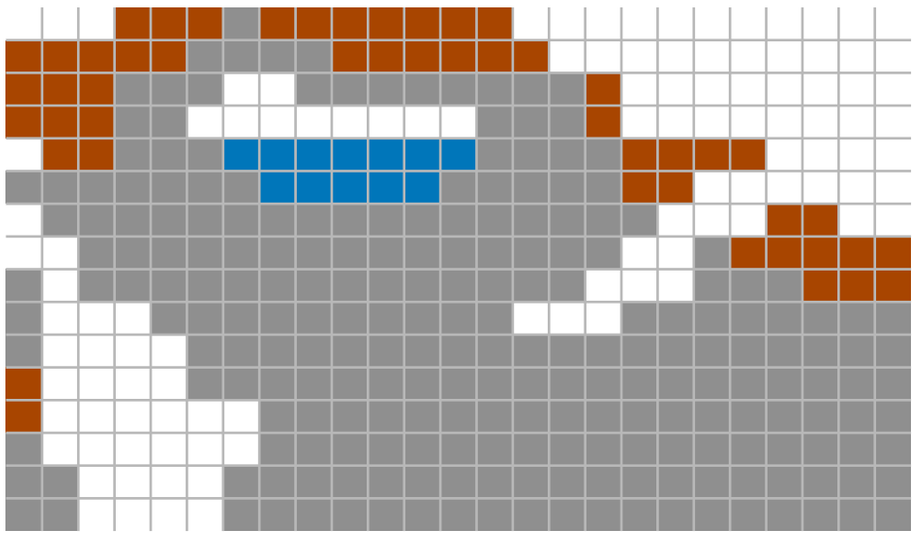
Player1

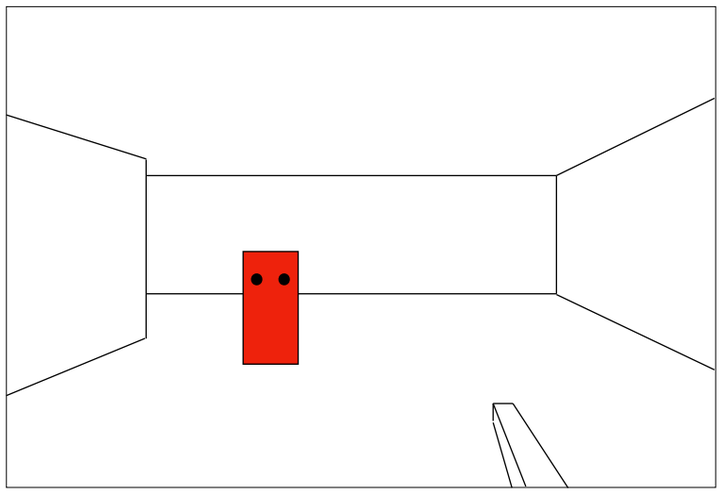
移動: 矢印 撃つ:p 置く: ;

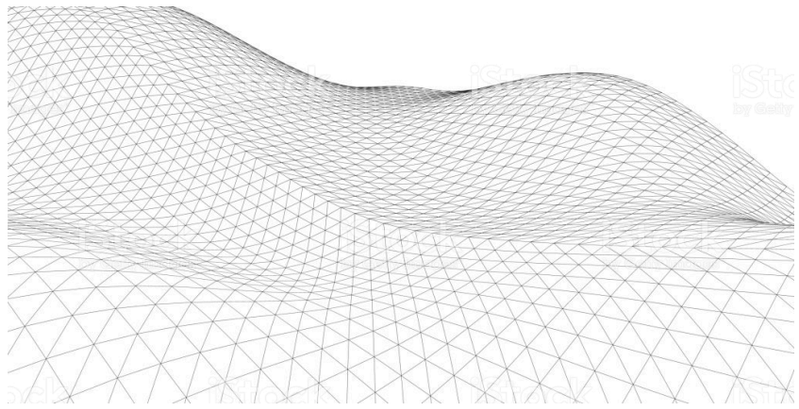


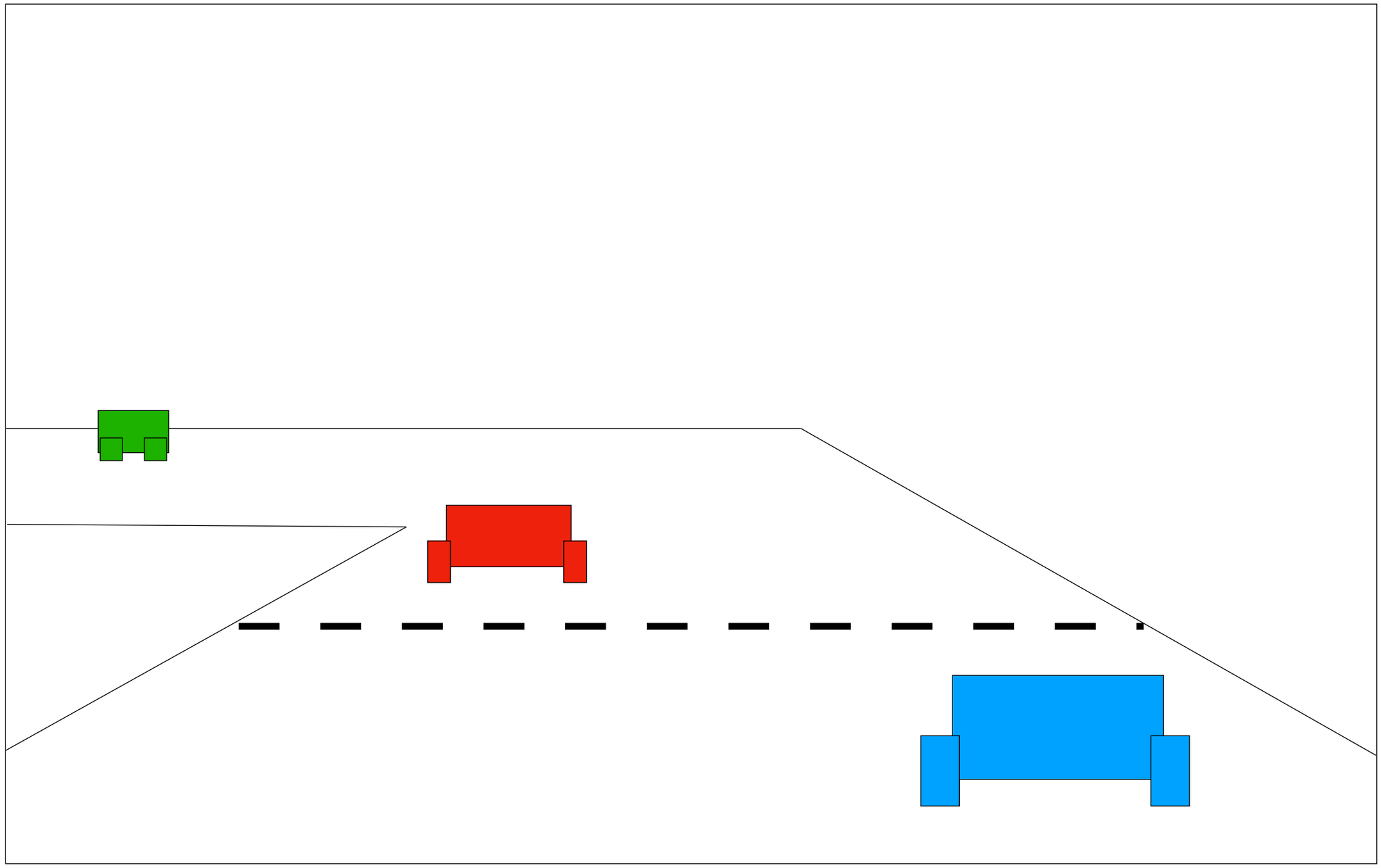


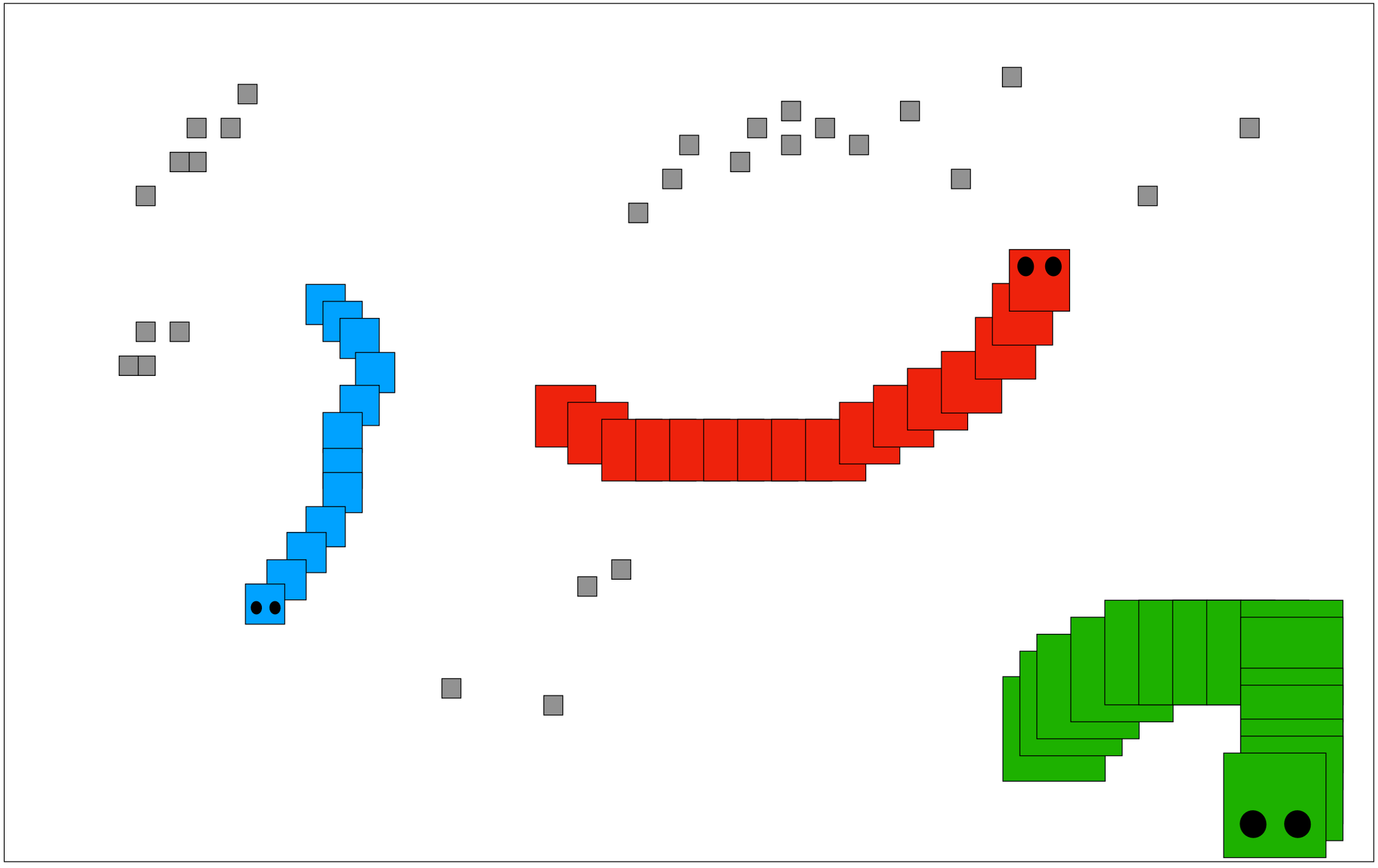


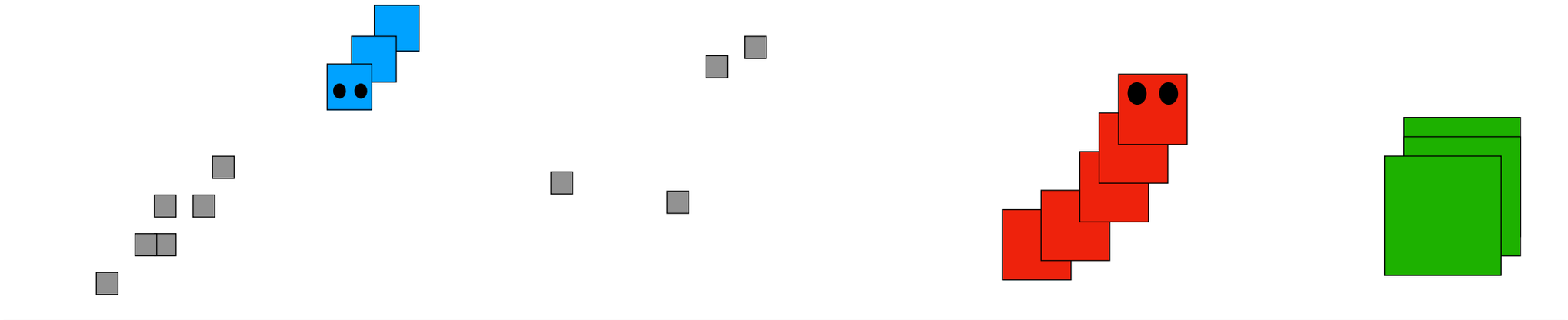


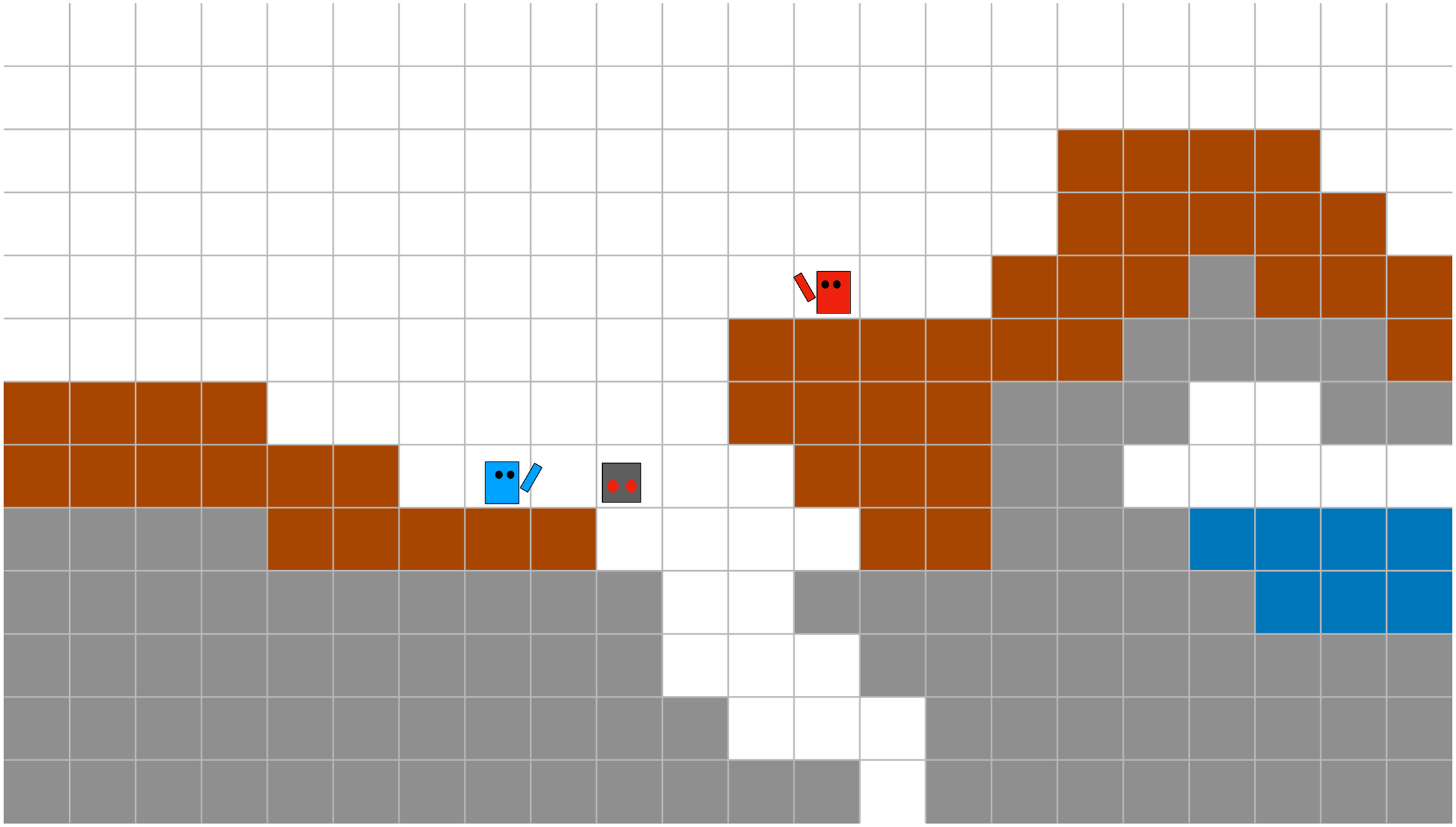




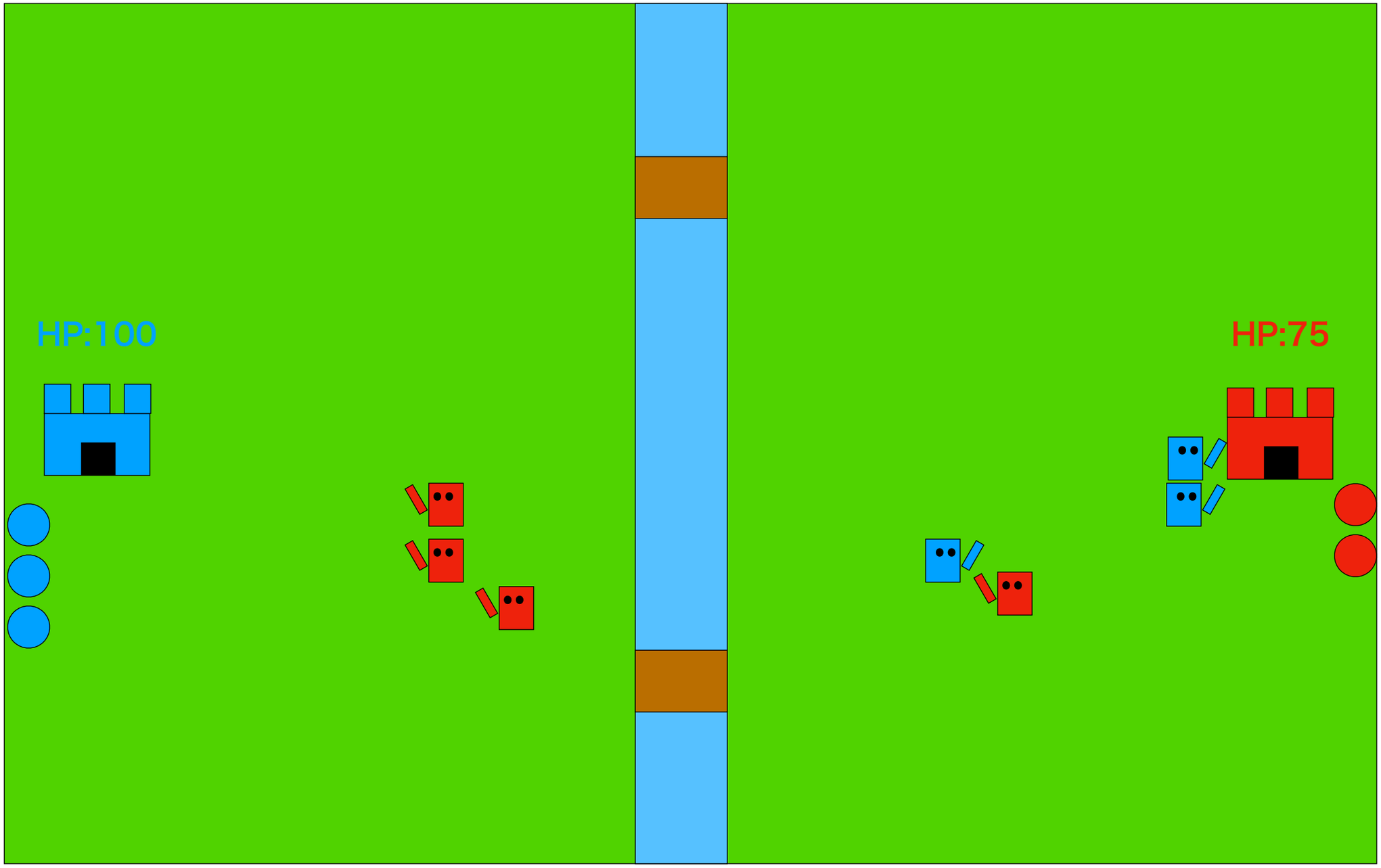






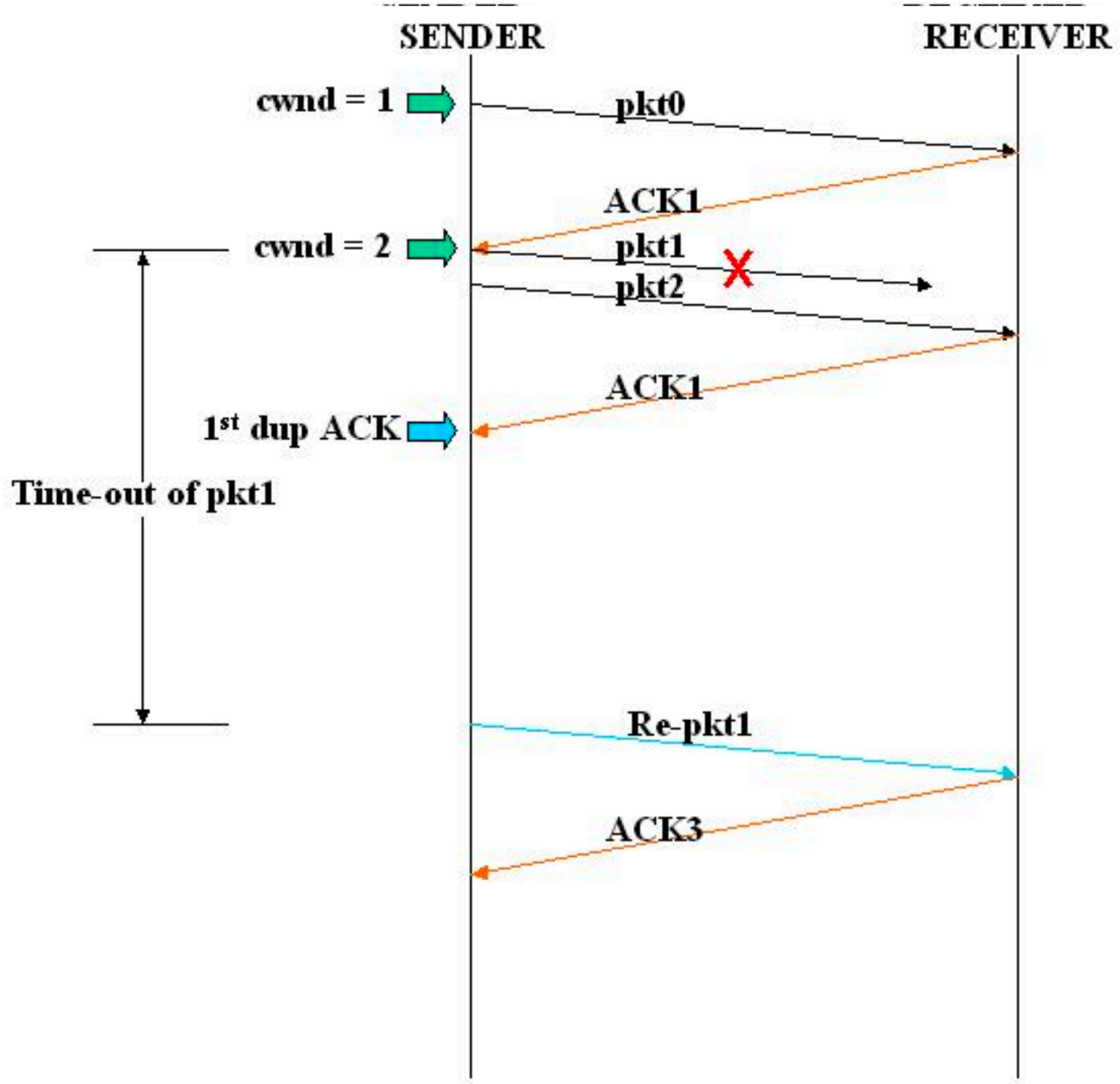


voxel.js * blocks in yo brows
voxel debris
substack.net/projects/voxel-forest/
To hatena
QRコード
ニコニコチャンネルツ
Press This
Game on HTML5
HTML_CodeSniffer
Visual Event
Pin It
日本語化
deCSS3
GistDeck
その他のブックマーク
58 FPS (0-60)

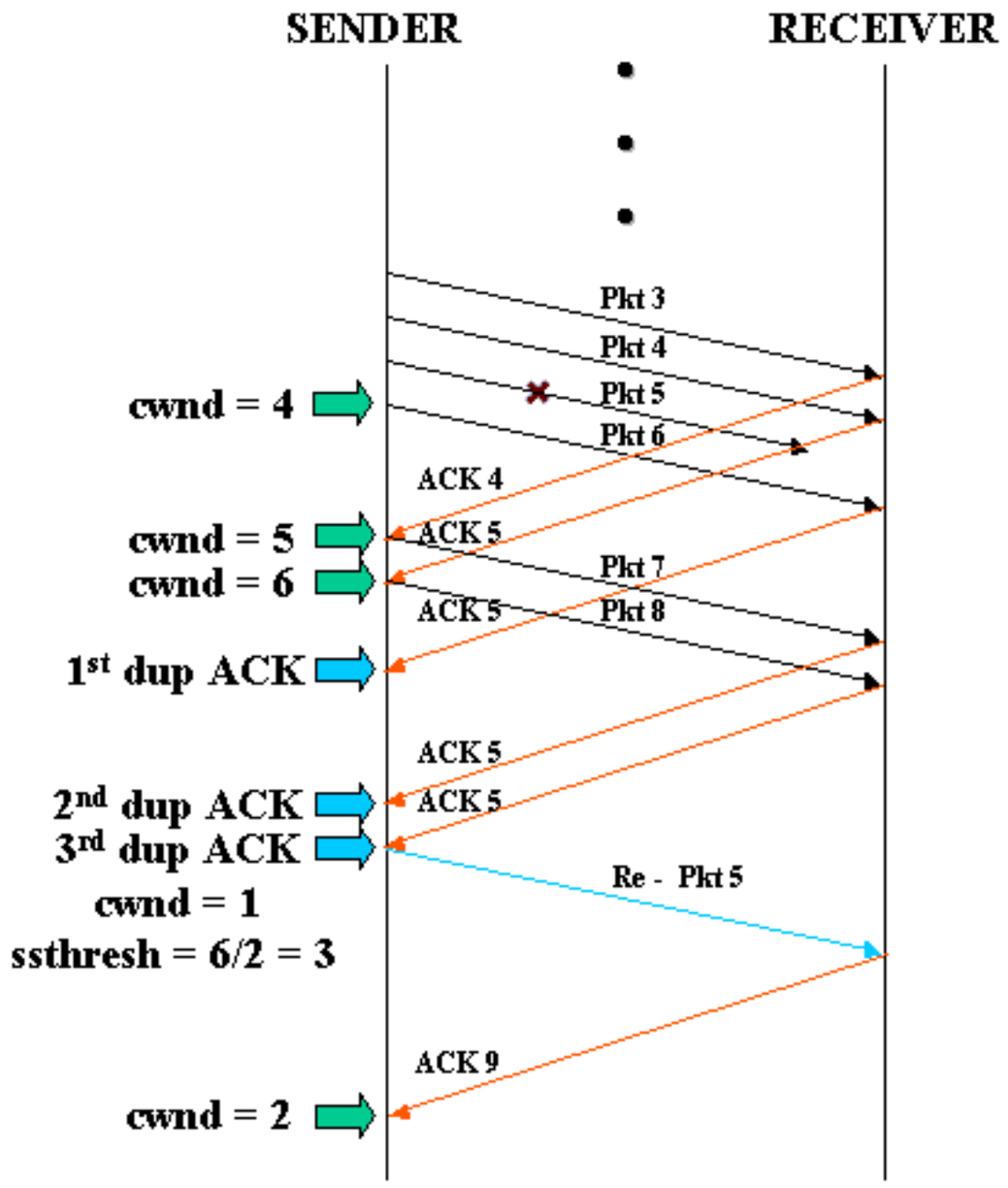

HP:100
HP:75

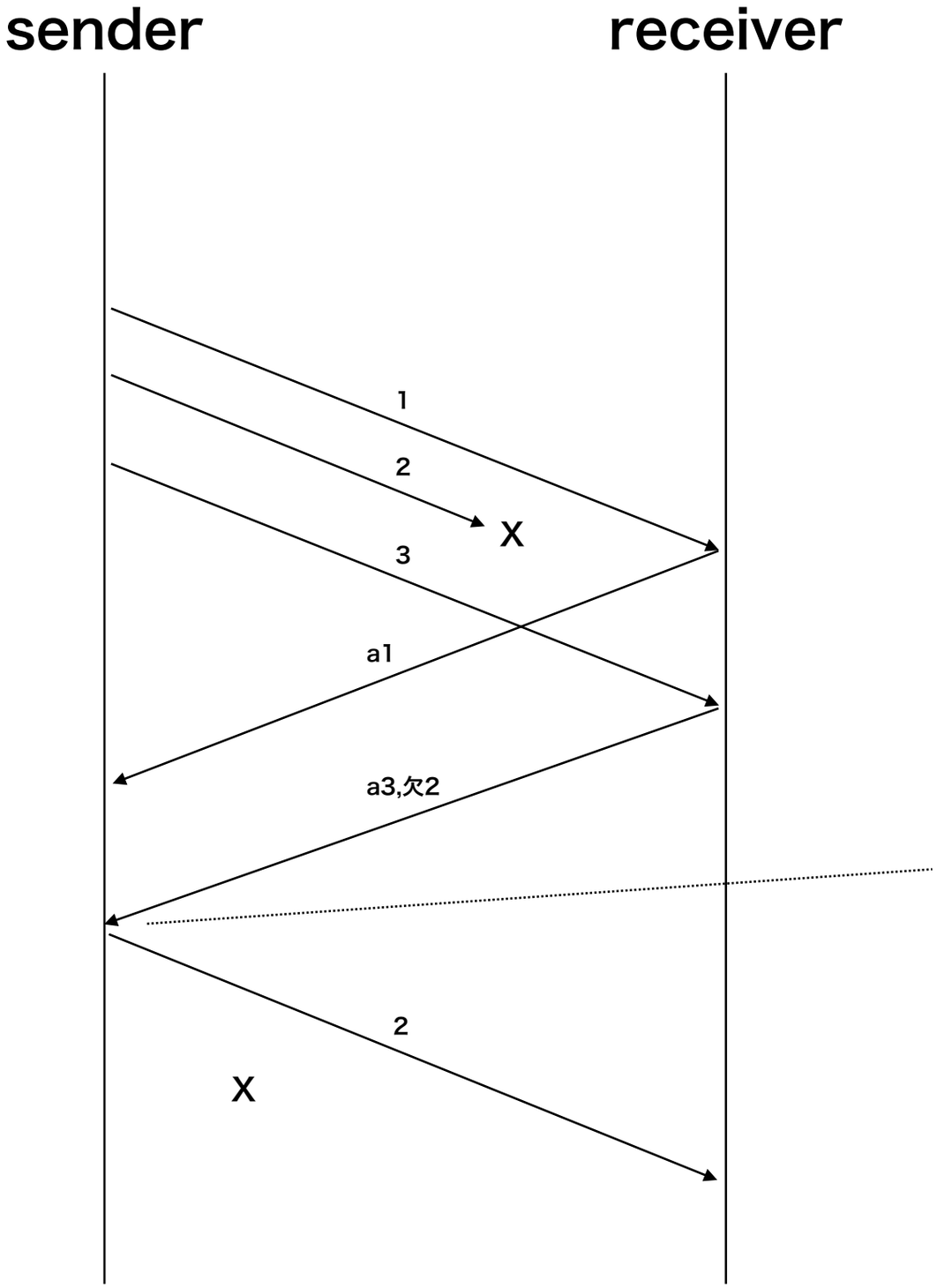
TCP duplicate ack

https://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_WANIDA/DR/JavisInActionFastRetransmitFrame.html



TCP fast retransmit



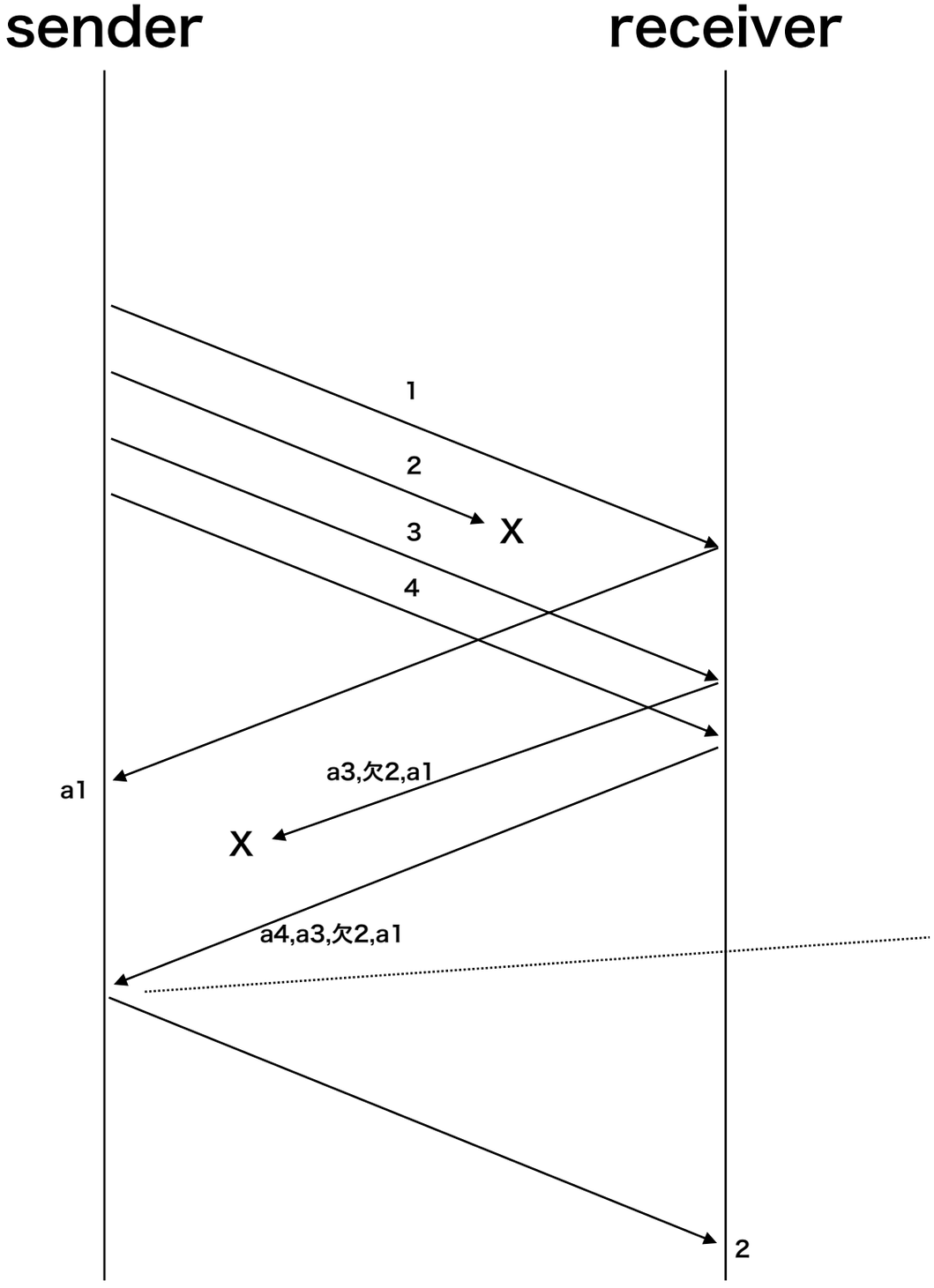


DCCP (RFC 4340) における ack vector

TCPのACKに、追加として過去N個分のパケットの着信状態を配列にして返す。
QUICでは、欠損しているパケットの番号リストを返す。

ack vector

ここで送信できれば、送信側でのタイムアウトや
dup ackが3つ重なることによる再送よりも早くできる場合がある。
しかし、a3が到着したときに、a2のackがまだ到着していないことは送信側でわかるのだから、
この2の再送は可能。しかし、



DCCP (RFC 4340) における ack vector

TCPのような単純なACKに、追加として過去N個分のパケットの着信状態を配列にして返す。
QUICでは、欠損しているパケットの番号リストを返す。

ack vector

ackがなくなっても、欠損リストがあれば、すぐ2を送れる。