Extending Client Side Prediction Methods in Networked Multiplayer First Person Shooter Games to Include Level Information

Richard Steele

***Abstract*—This paper critically examines the effectiveness of the current methods of client side prediction in networked multiplayer digital games, specifically first person shooter games where the accuracy of each game agent's behaviour is critical to the player's experience. The de facto use of *dead reckoning* is accepted as a standard tool to help minimise the number of required network updates, but its results can be error prone which are a detriment to the player's experience. This paper looks at the currently used dead reckoning algorithms and suggests using further data about the game state to provide a more accurate estimation of a game agent's location and behaviour at a future time. Information regarding the layout of a level is used influence acceleration factor in the dead reckoning algorithm and to dynamically alter the allowance threshold to better reflect an agent's behaviour at critical locations in the level by prompting an increased rate of player state updates.**

I. Introduction

WITH modern networked multiplayer digital games where the game state must be updated often, it is impractical to send and receive network updates on every frame. Instead, *dead reckoning* algorithms are used to predict future states of the game, and network updates are needed only when the actual behaviour differs from the predicted behaviour by a threshold amount. This idea of an acceptance threshold accepts that the perception of an entity for all other clients on a network may differ to the actual state of that entity. While effective at greatly reducing the required network updates and reducing the impact of network transmission delay [1], impossible behaviours such as tunnelling (projecting a position past a barrier), or improbable behaviours counter intuitive to the game are possible. This paper proposes that by using further information about the game state, such as level layout, an agent's behaviour can be predicted with greater accuracy to help minimise unwanted behaviour.

Previous papers in the context of games describe dead reckoning as a measure to account for network latency compensation. Few expand beyond this and are concerned generally with how best to marry the predicted state with the actual state by interpolation or rolling back. This paper explores improving the accuracy of the simulation by dynamically adjusting the threshold allowed by traditional dead reckoning methods to prompt more frequent state updates at critical times when the greatest accuracy is required.

It should be noted that not all networked multiplayer games suffer from latency problems. For instance turn based games consider each client sequentially. Where there is no scope or benefit for a player to react immediately, no measures must be taken to account for problems incurred by some milliseconds lost in transmission. These issues are prevalent in fast paced competitive games such as racing games, or the first person shooter genre. This paper then bases its study upon the first person shooter game genre, sometimes referred to as *twitch* shooters [2]. The need to respond quickly to other clients' perceived actions is critical to the player experience, therefore it is crucial that those actions are shown as accurately as possible.

The research question that this paper addresses is: how can client side prediction in networked multiplayer first-person shooter games that use dead reckoning be improved?

II. Background

This literature review first confirms the definition of client side prediction, the problems that it solves, and the trade off against absolute accuracy. Dead reckoning is explained and explored with a focus on the real world applications and techniques of its use within games. Finally, alternative methods of solving the the problems associated with network latency are explained and acknowledged.

A. Client side prediction

Networked multiplayer games are defined as sharing a game space between multiple clients [3]. Each client holds a representation of that game space which it uses to display information to the user, and allows the user to change within the rules of the game. When elements or details of the agents of that game space are changed by a client's actions, the client assumes that the action is successful, performs that action on its own game state representation, then sends the action information to the network so as all other clients can update their own game space representation to reflect the performed action [4].

Problems can occur when:

- the updated information is received too late (network latency)
- the updated information is not received (packet loss)
- the action is not successful (simultaneous conflicting actions are performed by many clients).

Network latency forms the primary bottleneck for online game performance with an acceptable value being under 100 milliseconds [2] [5] and worst case overall latencies claimed to be up over 1 second [6]. To counter this, each client takes the slightly old network state update that it receives and

brings it approximately up to date through extrapolation before displaying it to the user. Performance degrades significantly under variance in latency (or *Jitter*) [7] [8]. The jitter effect must be continually accounted for as the delta time value between updates can be subject to change. As such, the current time in which an action is performed is included with the update packet of data sent and received by clients, and forms an integral part of the prediction algorithm. The latency time difference must be continuously evaluated by sending and receiving time stamped packets to maintain a correct delta time value [9], however manipulating these time stamps is an opportunity for cheating [10] so must be secure from interference or if using a centralised server then a globally synchronised clock can be used [11].

Given the vast quantity and unreliable nature of the messages being sent [12], packet loss must be planned for. The same dead reckoning methods used to account for network latency, and outlined in *B.* can be used to replace missing packets of information, or application data units (ADUs) [3] sometimes referred to as protocol data units (PDUs) [13], to supply a prediction of where the ADU's agent will "most probably" be at a given time.

B. Dead reckoning

Dead reckoning methods are based on a navigational technique of estimating one's position based on a known starting point and velocity [5]. In the simplest implementation, we take the last position we received on the network, and project it forward in time with at its last known velocity [14].

Fig. 1. First order linear projection. Image sourced from [14]

This simple representation of projecting forward is correct if an agent has linear movement and constant velocity. However, agents in a first person shooter game often have complex non linear movements which will need to be checked for and accommodated. Real world applications of linear dead reckoning also suffer with having to account for unpredictable behaviour e.g. drift due to wind or wheel slippage [15] [16]. The techniques used for managing these behaviours are transferable to the problems of dead reckoning networked multiplayer game agents with complex behaviour.

In 1983, with substantial support from the U.S. army, the Defense Advanced Research Projects Agency (DARPA) initiated the Simulator Networking (SIMNET) program which developed dead reckoning techniques to support a virtual world to train soldiers [17]. Their approach to networking solutions has been incorporated into the Distributed Interactive Simulation (DIS) standard which is now an IEEE standard [13]. It outlines the use of a PDU to help dead reckoning handle complex behaviour [18]. It is worth remembering however that DIS was designed for use by a small number of players (less than 50), and the inclusive PDU model can be criticised for holding too much redundant data [19].

Each client maintains a dead reckoned model of itself that corresponds to the model used by other clients to "see" its represented agent. The client must regularly check whether the difference between the predicted state calculated with dead reckoning that all clients share and the actual state exceeds a certain threshold. If this is the case, the client is then responsible to generate an updated entity state for the dead reckoning algorithm to apply to the model. Then send that updated state to all clients on the network to learn about the correct new state of the entity to then use to update their own model of that client's agent [17] [20]. This also happens when a fixed amount of time has passed since the last update (normally 5 seconds) [21].

Fig. 2. Image sourced from [22]

Were this updated information used immediately, the modelled entity would appear to jump (fig. 2). Instead, the dead reckoning algorithm uses interpolation and projective velocity blending [14] to tend towards and eventually reconcile its own dead reckoned model with the updated state information.

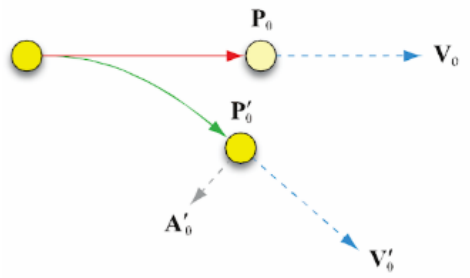

Fig. 3. Upon update, the difference between the simulation shown in red and the actual position shown in green is observed. Image sourced from [14].

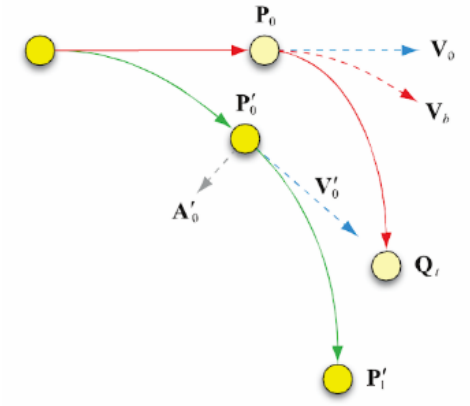

Fig. 4. With interpolation the simulation attempts to reconcile itself with the updated position and motion as explained in section C. Image sourced from [14].

C. Implementation

Reconciling deviations with projective velocity blending

$$Q_t = P_0 + VT \tag{1}$$

$$Q_t = P'_0 + V'_0 T + \frac{1}{2} A'_0 T^2 \tag{2}$$

$$V_b = V_0 + (V'_0 - V_0)\hat{T} \tag{3}$$

$$P_t = P_0 + V_b T_t + \frac{1}{2} A'_0 T_t^2 \tag{4}$$

$$P'_t = P'_0 + V'_0 T_t + \frac{1}{2} A'_0 T_t^2 \tag{5}$$

$$Q_t = P_t + (P'_t - P_t)\hat{T} \tag{6}$$

Where Q_t is the dead reckoned position at a specific time T for an entity starting at position P, moving at velocity V, with acceleration A. As shown in fig 3 and fig 4, prime ($'$) refers to the actual entity rather than the simulated model. The subscript 0 refers to the starting value, subscript t is that value at time T, and subscript b is the blended model between actual and simulation values.

(1) First order linear projection. Shown in fig 1 as projecting a velocity over time from a position.
(2) Second order projection adds the derivative of acceleration applied according to Newton's laws of motion.
(3) Shows a velocity blending model given two velocities. In client side prediction this would be the simulated velocity and the actual velocity. V_b will tend towards the actual velocity V'_0 as delta time $\hat{T}$ tends towards 1.
(4) Applies the blended velocity value with the simulation's starting position to the second order projection in (2) giving the simulated position at time T.
(5) Applies the last known actual values to the second order projection in (2) giving the updated projection position at time T.
(6) Blends these positions tending the simulation towards the updated projection position as delta time $\hat{T}$ tends towards 1.

THRESHOLD BASED DEAD RECKONING PSEUDO CODE

Input:
prediction error threshold: η
time at which the last position was sent: t_o
actual position (at time t): x_t
foreach *frame (time t)* **do**
  $\hat{x}_t \leftarrow$ prediction(x_{t_0}, t);
  if $||x_t - \hat{x}_t|| > \eta$ **then**
    send update packet(x_t)
    $t_0 \leftarrow t$
  end
end

Algorithm 1: A client predicts the position of their own agent for the current frame based on the last sent position update. If the prediction error is larger than the threshold η then an update packet is sent to the network and the simulation resets itself.

D. Interest modelling

The shortcomings of estimating a game agent's behaviour based solely upon motion dynamics are already recognised and attempts to address these problems with further information have been somewhat explored. Amir Yahyavi *et al* successfully use identified points of interest to influence their dead reckoning algorithm, improving the accuracy by up to 30% over traditional dead reckoning techniques [23]. They propose the use of *attraction forces*, derived from the player state and nearby points of interest, to be factored into the dead reckoning calculation alongside the current acceleration. The example that they often cite states that a player controlling an entity with reduced health will most likely move towards a health pack if there is one in sight. However, their method titled *AntReckoning* must decide on the attraction forces of each point of interest, to build a *pheromone map*. AntReckoning's weakness is great complexity as every change in the game state affects nearby attraction forces. Also the reliability of AntReckoning can be disputed as different players may have differing goals, but the increase in accuracy reported is excellent. This can also be used to influence the path finding algorithms of non playable characters (NPCs) controlled by an artificial intelligence to direct them towards popular parts of the level [24] [25]. While this shows an interesting application of utilising level information, the behaviour of NPCs is beyond the scope and focus of this project.

The merit of Amir Yahyavi *et al*'s work [23] [24] recognising and proposing solutions to the problems of solely using traditional dead reckoning methods in games with client side prediction is acknowledged, and an inspiration for this project.

E. Adapting the acceptance threshold

As shown in algorithm 1, the condition that prompts a network update is that the difference in position between a client's own entity and the representation of that entity used by the network differ by more than a threshold amount. This amount can be visualised as an acceptance radius around the dead reckoned position as shown in fig 2. Generally, dead reckoning algorithms employ a fixed threshold [26]. Related work that investigates changing this threshold amount depending on game state conditions decide the variance by considering the individual peer to peer relationships such as the distance between the entities. This *relevance filtering* is reliant upon a *radius of interest* [27]. The reasoning given is that distant entities are not likely to interact with each other so accuracy is not extremely important [26] [28].

This paper acknowledges this method's effectiveness in specific situations to reduce network updates. But, given the often edge to edge viewing range prevalent in many game first person shooter levels [29] [30] [31], this paper argues against its use in a general approach to improving client side prediction in first person shooter games.

To allow for edge to edge sight lines and the prevalent long range weapons available for use in first person shooter games, we can consider filtering for entities that are within that player's potential field of sight [12]. However, this requires previous knowledge of all entity positions therefore a two pass

approach would be needed to determine which entities would be affected by a state change, and then to update those clients. Also, care must be taken as even where the entities are out of sight of each other, dead reckoning may predict a behaviour which results in the entities being revealed and the integrity of the game state will be compromised.

Roberts *et al.* propose that network update time should be factored into deciding the threshold limit value. Network limitations when presented with a large amount of packets may result in inconsistency for which additional updates are needed to rectify [32]. This shortcoming in a networked application is an important consideration. However, given the local prediction methods of this project, fall outside of this study. Were future work to lead to a networked experiment, and given the proposed increase in packets sent at critical junctures, the relevance of considering network shortfalls when calculating a dynamically changing threshold will be crucial.

F. Doppelgangers

Doppelgangers offer a promising alternative that will adhere to the game's rules and avoid impossible behaviour. Jeffrey Pang *et al.* discuss populating a client's game state representation with computer-controlled players known as *bots*. Updates prompted by a threshold based system similar to that outlined in algorithm 1 will contain *guidance* for the artificial intelligence (AI) to rectify and better replicate the relevant representation [33] [34]. As most first person shooter games contain AI routines for single player modes, much of the work needed to guide a bot around a level has been done. Problems will occur if a bot representation decides that it will perform an action such as shoot at an enemy, whereas the actual behaviour does not. Questions arise such as should the bot then fire "blanks", and consideration must be given as to whether that action may inadvertently alert enemies to that entity's position? This approach shows considerable potential in providing possible and realistic representations of other clients' behaviours, but relies heavily upon accurately portraying and dynamically adjusting the player's behaviour model. To configure the AI behaviour, neural networks can be trained and used to predict changes of the entity velocity [35]. Jeffrey Pang *et al.* propose that a blending between an AI bot for non-critical behaviour e.g. outside of the client's focus, and conventional dead reckoning state updates for representations of entities within the player's focus will provide a satisfying result [33].

This paper argues in favour of bot representations as both a method to decrease the frequency of network updates, and as a tool to extrapolate future behaviours. But, also acknowledges its dependency upon accurate player modelling techniques. Player modelling is an ongoing study in the field of games. Since the focus of this work is on dead reckoning in client side prediction we omit further details about AI modelling player behaviour, although we strongly consider this as future work in the field of client side prediction.

III. Discussion

The two part purpose of client side prediction is to maintain an up to date game state for clients in the case of late or lost packets, and to use prediction algorithms to reduce required network updates. While the simplest dead reckoning algorithm provides a linear vector approximation based on velocity, we have shown above that interpolation and projective velocity blending can be used to smooth trajectories between state updates tending towards the correct representation. It is a fundamental requirement for believable games that the state transitions are smooth and believable, which is why curves are employed rather than a jump. These curves are shown to be accurate when inferred from polynomial modelling based on the Lagrange or Taylor series [36]. But require multiple states from which to extrapolate an approximation. Packet loss can be greatly detrimental to this approach. Small sample sizes combined with potentially erratic movement of game players will likely produce unlikely results. A solution would be to provide back up copies of previous states in each PDU to restore missing information. A technique already employed in Quake where the last three commands are also sent in each packet to compensate for lost packets [12]. James M. Van Verth and Lars M. Bishop provide a method to sample and project cubic bezier spline curves [37] but these are shown to be slightly less accurate than the velocity blending technique shown above proposed by Curtiss Murphy [14].

Dead reckoning is an *optimistic algorithm*. It assumes it is correct and when it is not, it must adjust itself. It is apparent that the blending of this adjustment may result in compromising behaviour. Demonstrated in first person shooter games as appearing to other players in a vulnerable position outside of cover, or moving through an impenetrable area as defined by the game's movement rules. Consider the velocity blending method shown in fig 4 and (6). A player could turn instantaneously and the algorithm would apply unrealistic forces to create huge accelerations at arbitrary angles [4]. Large inconsistencies due to extremely late updates may even lead to repairs requiring a state rollback which will cause more drastic position jumps [38]. The resulting movement shown by the red line in fig 4 may fall outside the rules of the game. Either passing through untraversable locations such as through walls or over gaps, or into unlikely locations according to the game's rules such as into a vulnerable position outside of cover. Many networked multiplayer first person shooter games suffer from criticism because of this. Players can feel unfairly treated and forums discussing these issues become very popular with many thousands of views [39] [40] [41] [42] [43] [44].

The de facto method of dead reckoning entities' behaviours is a distributed approach that does not require a centralised server. Using a distributed approach is mandatory for many DVEs in order to avoid the well known problems of centralised systems such as increased latency, single-point-of-failure and lack of scalability [20]. However, having a dedicated central authoritative server (or a single client nominated as such) means that even if a client simulates different results than the server, the server's results will eventually correct the client's incorrect simulation [4].

second order derivatives are sufficient for short-term dead reckoning [23].

Similar to in a racing game where a player will slow down

before a corner [45]

Laurent Gautier and Christophe Diot propose that late packets due to network latency may be accommodated by the client by introducing a local presentation delay to provide a consistent, distributed state [46]. Trailing state synchronisation is designed specifically for real-time multiplayer games, and achieves better responsiveness and scalability while maintaining near-perfect consistency [12]. However, these increase the application-level delay even more so. While this method termed *time warp* is acceptable for their MiMaze maze navigation project, it is unsuitable for the fast paced style of first person shooter games.

In the interest of completeness, in $II.A.$ we mention network latency not succeeding due to a state conflict, where simultaneous conflicting actions are performed by multiple clients. In this case, the authoritative server, the client acting as such, or the game itself, must decide which state is the "correct" state and update all conflicting clients with the decision. Games will often invoke a small delay before critical moments e.g. a death animation, to allow conflicting states to resolve themselves before committing to a particular irreversible state. With this delay, the game can rollback to the correct state without the user experiencing a jarring state change [20].

A. Representing level data

As common to most games, AntReckoning assumes a game world divided into non overlapping cells, e.g., Delaunay triangulation, Voronoi tessellation, binary space partitioning, or regular grids (e.g., square grid in Figure 1)

[47]

[48]

[49]

[50]

[51]

map partitioning [52]

Finally, to visualise the level data that this project uses to adjust it's client side prediction request rate...

to aggregate PDU states at critical junctions

Using game state information to adapt client side prediction is still in its infancy.

---

the *Donnybrook* protocol [34] looks to increase the delay between packets up to seconds by leveraging human cognitive limitations to preferentially spend bandwidth capacity on information most important to the player's *interest sets*.

IV. Evaluation

The literature shows a broad approach to the trade off between network latency implications and accurately representing every entity's behaviour. Finding the balance between excessive network congestion and maintaining an accurate representation of the game state for all clients is a challenge faced by all networked multiplayer games. Dead reckoning methods are highly effective, but are solely concerned with an entity's predictable motion. Derivatives such as AntReckoning attempt to model a player's likely intention, although extending this player modelling to guide a bot representation can be rife with inaccuracy. It is apparent that where an assumption of a player's actions are incorrectly predicted, rolling back the game state can be a jarring and detrimental experience to the users.

Where network latency would cause the received data to be slightly behind, client side prediction algorithms like dead reckoning can prolong the interval of message transmissions and abolish the network latency at the cost of data consistency. In the case of lost packets resulting in no update data for that agent, client side prediction methods can be used to simulate an agent's behaviour reducing the impact of a network transmission delay. When used effectively this can help to synchronise the game state across all clients.

Related work seeks to minimise the number of required updates and maintain an up to date game state for all clients despite network latency and jitter. However, since the DIS was adopted as an IEEE standard in 1998, network bandwidth has increased considerably. This paper proposes that less strict measures be taken to reduce network load, contrasting the approaches of current methods. This paper encourages increasing the rate of network updates to be sent by dynamically altering the acceptance threshold of the client side prediction method when an entity is in a critical location in a game. Altering the threshold has been explored before, as has location evaluation in the form of interest modelling or by area of interest, but to this paper's knowledge no study has combined altering this threshold amount by a predetermined vector map showing locations where changes in motion are most unpredictable.

Furthermore, changes in acceleration shown to be common in the entity's location can be combined with the last known acceleration value in the second order projection of motion ($C.$ (2)) in a similar manner to how AntReckoning combines attraction forces. This will attempt to preempt changes in acceleration, not only in the entity's forward direction, but in the direction suggested by the vector map.

Unfavourable reactions from users about poor game state management show the importance of finding an effective solution to minimising client side prediction error and forms the motivation for this project.

V. Methodology

It can take some time to get used to controlling an entiity in a first person shooter.

All participants who went outside of the track 9 or more times during the 9 laps (not including the warm up lap) were excluded from the experiment. This resulted in 3 out of the 26 participants being excluded. The reason why very poor drivers were excluded was to not introduce noise into the statistics or include data from some one who wanted disrupt the experiment on purpose. Some of the fastest laps were completed in just over 16 seconds and some of the slowest, in about 30 seconds. It could have been good to also set a limit on how long a race was allowed to take before it was excluded. However, the slowest laps were already excluded because of the first criteria. Since no participant was observed deliberately driving slow or taking breaks during the experiment, a time limit was never introduced.

As a reference to see how great the distances error would be if no prediction was performed, an algorithm that did not do any updates or prediction was implemented. The data read from the outdated node is simply passed on as the predicted position.

This method takes the outdated position and adds the velocity multiplied by the latency. This way a linear prediction is made.

A. Hypothesis

B. Hypotheses:

Table I shows the hypotheses that are tested in this experiment. The methodologies that are used to test these hypotheses are play-testing and a questionnaire. As this required human participants for both parts ethics approval for this methodology was gained from Falmouth University's Research Ethics Board.

C. Participants

REFERENCES

[1] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Proceedings of the 1st workshop on Network and system support for games*. ACM, 2002, pp. 79–84.

[2] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, and J. Flinn, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 151–165.

[3] C. Diot and L. Gautier, "A distributed architecture for multiplayer interactive applications on the internet," *IEEE network*, vol. 13, no. 4, pp. 6–15, 1999.

[4] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Game Developers Conference*, vol. 98033, no. 425, 2001.

[5] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," *The Electronic Library*, vol. 20, no. 2, pp. 87–97, 2002.

[6] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[7] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003®," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 144–151.

[8] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–7.

[9] J. Glazer and S. Madhav, *Multiplayer game programming: Architecting networked games*. Addison-Wesley Professional, 2015.

[10] E. C. B. F. S. Jamin, "Cheat-proofing dead reckoned multiplayer games," *Ann Arbor*, vol. 1001, pp. 48 109–2122, 2003.

[11] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 161–165.

[12] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," in *University of Michigan*. Citeseer, 2001, pp. 3–6.

[13] D. S. Committee *et al.*, "Ieee standard for distributed interactive simulation-application protocols," *IEEE Standard*, vol. 1278, pp. 1–52, 1998.

[14] C. Murphy, "Believable dead reckoning for networked games," *Game engine gems*, vol. 2, pp. 307–313, 2011.

[15] H. Chung, L. Ojeda, and J. Borenstein, "Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 1, pp. 80–84, 2001.

[16] L. Ojeda, G. Reina, and J. Borenstein, "Experimental results from flexnav: An expert rule-based dead-reckoning system for mars rovers," in *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, vol. 2. IEEE, 2004, pp. 816–825.

[17] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The simnet virtual world architecture," in *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*. IEEE, 1993, pp. 450–455.

[18] W. D. McCarty, S. Sheasby, P. Amburn, M. R. Stytz, and C. Switzer, "A virtual cockpit for a distributed interactive simulation," *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 49–54, 1994.

[19] T. Henderson, "Latency and user behaviour on a multiplayer game server," in *International Workshop on Networked Group Communication*. Springer, 2001, pp. 1–13.

[20] M. Mauve, "How to keep a dead man from shooting," in *International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Springer, 2000, pp. 199–204.

[21] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," University of Delaware, Tech. Rep., 03 1992.

[22] J. Arronson. (1997) Dead reckoning: Latency hiding for networked games. [Online]. Available: http://www.gamasutra.com/view/feature/131638/dead_reckoning_latency_hiding_for_.php

[23] A. Yahyavi, K. Huguenin, and B. Kemme, "Antreckoning: dead reckoning using interest modeling by pheromones," in *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2011, p. 1.

[24] A. Yahyavi, B. Kemme, and K. Huguenin, "Interest modeling in games: the case of dead reckoning," *Multimedia systems*, vol. 19, no. 3, pp. 255–270, 2013.

[25] A. Yahyavi, J. Tremblay, C. Verbrugge, and B. Kemme, "Towards the design of a human-like fps npc using pheromone maps," in *Games Innovation Conference (IGIC), 2013 IEEE International*. IEEE, 2013, pp. 275–282.

[26] W. Cai, F. Lee, and L. Chen, "An auto-adaptive dead reckoning algorithm for distributed interactive simulation," in *Proceedings of the thirteenth workshop on Parallel and distributed simulation*. IEEE Computer Society, 1999, pp. 82–89.

[27] S. J. Rak and D. J. Van Hook, "Evaluation of grid-based relevance filtering for multicast group assignment," in *Proc. of 14th DIS workshop*. Citeseer, 1996, pp. 739–747.

[28] I. Jaya, E. S. Liu, and Y. Chen, "Combining interest management and dead reckoning: a hybrid approach for efficient data distribution in multiplayer online games," in *Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on*. IEEE, 2016, pp. 92–99.

[29] TobysCS. (2014) Cs:go map callouts (overviews). [Online]. Available: https://www.tobyscs.com/csgo-map-callout-overviews/

[30] G. Ziervogel. (2014) The five greatest multiplayer maps of all time. [Online]. Available: https://www.nag.co.za/2014/01/27/the-five-greatest-multiplayer-maps-of-all-time/

[31] (2018) Call of duty: Black ops 4 multiplayer maps. [Online]. Available: http://callofduty.wikia.com/wiki/Category:Call_of_Duty:_Black_Ops_4_Multiplayer_Maps

[32] D. Roberts, R. Aspin, D. Marshall, S. Mcloone, D. Delaney, and T. Ward, "Bounding inconsistency using a novel threshold metric for dead reckoning update packet generation," *Simulation*, vol. 84, no. 5, pp. 239–256, 2008.

[33] J. Pang, F. Uyeda, and J. R. Lorch, "Scaling peer-to-peer games in low-bandwidth environments." in *IPTPS*, 2007.

[34] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 389–400, 2008.

[35] A. Mccoy, T. Ward, S. Mcloone, and D. Delaney, "Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 17, no. 4, p. 16, 2007.

[36] D. Hanawa and T. Yonekura, "A proposal of dead reckoning protocol in distributed virtual environment based on the taylor expansion," in *Cyberworlds, 2006. CW'06. International Conference on*. IEEE, 2006, pp. 107–114.

[37] J. M. Van Verth and L. M. Bishop, *Essential mathematics for games and interactive applications: a programmer's Guide*. CRC Press, 2008.

[38] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," in *Proceedings of the 1st workshop on Network and system support for games*. ACM, 2002, pp. 67–73.

TABLE I
HYPOTHESES

	Hypothesis	**Null Hypothesis**	**Data Collection Source**
1	Using bot behaviour in a level to build a vector map of aggregate changes in motion will form a representation of that level.	Visualising path-finding has no effect on the percent of the level the participant explores.	Game Data
1	Using bot behaviour in a level to build a vector map of aggregate changes in motion will form a representation of that level.	Visualising path-finding has no effect on the percent of the level the participant explores.	Game Data

[39] R. Cork, "Cs:go netcode issues," 2013. [Online]. Available: https://www.youtube.com/watch?v=bxJQ6KIoVCI

[40] (2014) Getting killed around corners. [Online]. Available: https://gamefaqs.gamespot.com/boards/713745-call-of-duty-ghosts/67947149

[41] Drift0r, "Black ops 2 in depth - lag compensation explained!" 2013. [Online]. Available: https://www.youtube.com/watch?v=xyCQtUFOJmA

[42] u/Solaire Lives. (2016) Dying round corners. [Online]. Available: https://www.reddit.com/r/Infinitewarfare/comments/5bci72/dying_round_corners_losing_every_gunfight_wtf_is/

[43] u/Soulstone_X. (2018) This is why people jump shot you around corners. did a little testing - offline with no lag. [Online]. Available: https://www.reddit.com/r/Blackops4/comments/9p1lit/this_is_why_people_jump_shot_you_around_corners/

[44] u/Her Protector. (2015) Getting killed through a wall or just after rounding a corner? [Online]. Available: https://www.bungie.net/en/Forums/Post/109909127

[45] E. Larsson, "Movement prediction algorithms for high latency games: A testing framework for 2d racing games," 2016.

[46] L. Gautier and C. Diot, "Design and evaluation of mimaze a multiplayer game on the internet," in *Multimedia Computing and Systems, 1998. Proceedings. IEEE International Conference on*. IEEE, 1998, pp. 233–236.

[47] C. Bauckhage, R. Sifa, A. Drachen, C. Thurau, and F. Hadiji, "Beyond heatmaps: Spatio-temporal clustering using behavior-based partitioning of game levels," in *Computational Intelligence and Games (CIG), 2014 IEEE conference on*. IEEE, 2014, pp. 1–8.

[48] B. Kuipers, "Modeling spatial knowledge," *Cognitive science*, vol. 2, no. 2, pp. 129–153, 1978.

[49] L. Wilkinson and M. Friendly, "The history of the cluster heat map," *The American Statistician*, vol. 63, no. 2, pp. 179–184, 2009.

[50] A. Moumtzidou, S. Vrochidis, E. Chatzilari, and I. Kompatsiaris, "Discovery of environmental resources based on heatmap recognition," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE, 2013, pp. 1486–1490.

[51] A. Drachen and M. Schubert, "Spatial game analytics," in *Game Analytics*. Springer, 2013, pp. 365–402.

[52] A. Steed and R. Abou-Haidar, "Partitioning crowded virtual environments," in *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 2003, pp. 7–14.