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 representing 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 to influence the 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 to minimise representation error.

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 informing the acceleration factor of the motion projection with the level's acceleration trend for the entity's location thereby tending it towards a norm, and 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.

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 need 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 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 multiple 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 client'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.
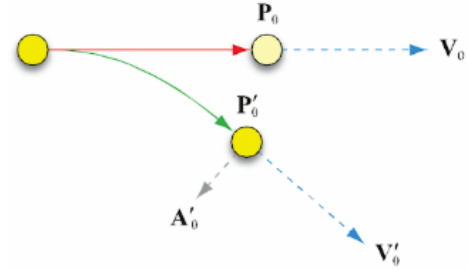

Fig. 3. Upon update, the difference between the simulation shown in red and the actual position shown in green is observed. Image sourced from [14].
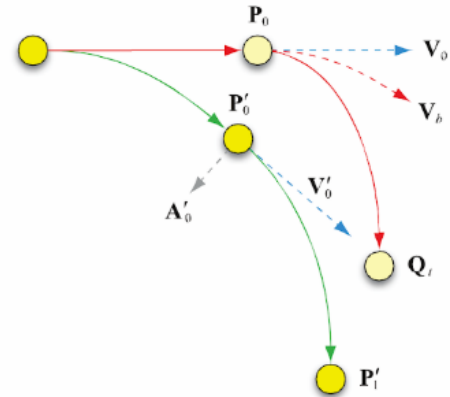

Fig. 4. With interpolation the simulation attempts to reconcile itself with the updated position and motion as explained in section C. Image sourced from [14].

C. Implementation

Reconciling deviations with projective velocity blending

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

$$Q_t = P_0' + V_0'T + \frac{1}{2}A_0'T^2 \tag{2}$$

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

$$P_t = P_0 + V_bT_t + \frac{1}{2}A_0'T_t^2 \tag{4}$$

$$P_t' = P_0' + V_0'T_t + \frac{1}{2}A_0'T_t^2 \tag{5}$$

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

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**
    $\hat{x}_t \leftarrow \text{prediction}(x_{t_0}, t)$;
    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. The *Donnybrook* protocol [23] 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*. Similarly, 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 [24].

Yahyavi *et al* 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 [25] [26]. 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 [24] [25] 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 [27]. 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* [28]. The reasoning given is that distant entities are not likely to interact with each other so accuracy is not extremely important [27] [29].

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 [30] [31] [32], 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 when 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 [33]. 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. Doppelgängers

Doppelgängers 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 [34] [23]. 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 bot's AI 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 [34].

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.

Benjamin Kuipars states that the difficulty of a dead reckoning task should vary according to the number of turns in the route, while the error rate and magnitude should vary according to the departure from straight [36]. So, we can infer that likewise when considering complex paths with many turns, to expect a larger error rate.

It is a fundamental requirement for believable games that the state transitions and corrections are smooth and believable, shown in figures as a curve tending towards the target rather than a jump. These curves are shown to be accurate when inferred from polynomial modelling based on the Lagrange or Taylor series [37]. 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 [38] 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 lead to repairs requiring a state rollback which will cause more drastic position jumps [39]. 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 [40] [41] [42] [43] [44] [45].

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 distributed virtual environments (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 if a client simulates different results than the server, the server's results can eventually correct the client's incorrect simulation [4].

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-B 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 that 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

This project proposes that information pertinent to the level layout can be used to better predict an agent's behaviour, and more critically evaluate when to send update packets. As such, effectively storing and evaluating spatial information about the changes in motion of agents moving around a level is paramount.

A heat map is a visual reflection of a statistical model [50] used often in games to graphically represent spatial analytics
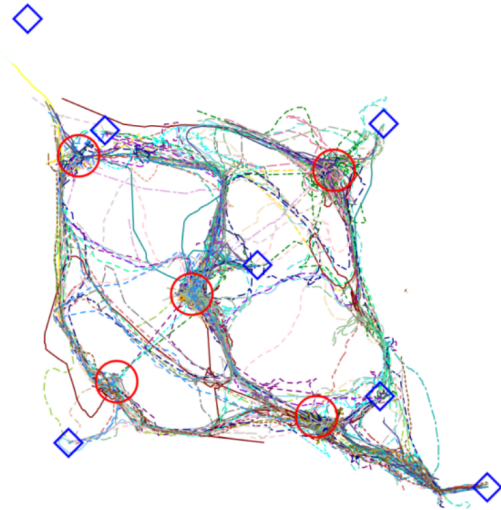
Fig. 6. 980 player trajectories from the "Arathi Basin" battleground from the game *World of Warcraft*. Red circles show strategic points and blue diamonds show graveyards which are used as spawn locations. Image sourced from [48].

Fig. 7. Representing spatial clustering and player movement in the "Gael" map from the game *Unreal Tournament 2003*. Red areas show clustering behaviour. Image sourced from [49].
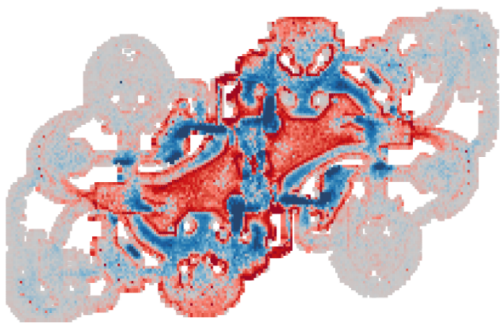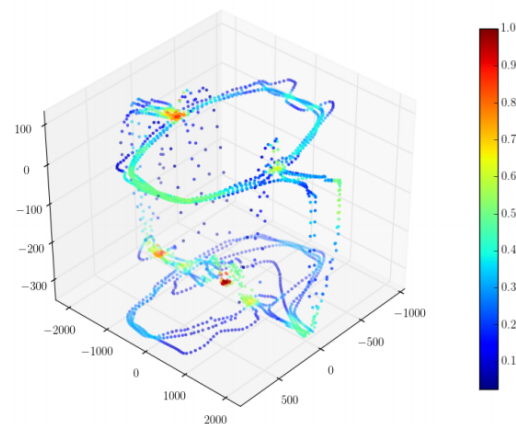

Fig. 5. Example of overlay analysis. A balance heatmap from the "Molten" map from the game *Transformers: War for Cybertron*. Red areas indicate deaths and blue areas indicate kills. Image sourced from [47].

of player data with colours, overlayed upon a traditional map of the level (see fig 5). They are a tool to help extract patterns and knowledge which can be used to detect outlying data [47] (Apparent in fig 6 as single paths travelling perpendicular to the trend). To this end, as a tool, it meets our needs perfectly. Where interest modelling seeks to better predict a single entity's interest, our proposed method outlines a broadly applicable standard per level by using irrefutable information about the level to better avoid dead reckoning methods projecting a client's position into an unlikely or impossible location. Outlying data, although still possible, represent a less likely path for the prediction algorithm, therefore predictions along that path should be considered with less confidence. A complete map of movement through a level would show all possible paths. Any dead reckoned path that did not adhere to that layout can be deduced to be acting impossibly, and thereby treated with a very low confidence or discredited entirely.

Unlike the path visualisation as shown in fig 7, the direction of motion must be asymmetric. Consider that a player may jump down from a platform but be unable to get back onto it [49]. The data must also be stored as such that symmetric paths such as corridors must preserve a direction vector, so as opposite paths do not cumulate to a zero score. Our method relies on the idea that a perpendicular vector to the trend must be realised as countering the norm.

Finally, the map must be partitioned in such a way that the information is relevant. Equal size cell partitioning can disproportionately accumulate several critical positions within, while other less dense cells may merely be a space of travel between key junctures [51]. It is impossible to declare equally important cell boundaries before collecting some data on the level so this project must rely upon extracting and analysing level data before formatting it into a state that the client side prediction algorithms can effectively use.

Heat maps are used in games for training bots to behave more realistically and improve map design during development [49] but to the best of our knowledge have not been applied to alter attraction forces in dead reckoning algorithms or alter the network update frequency.

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 when 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 at 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 (consider in a racing game, a player will slow down before a corner [52]) 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 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. While the focus of this project is on first-person shooter games, any improvement to client side prediction accuracy will be applicable to other networked multiplayer games that rely upon client side prediction and dead reckoning to maintain an up to date state representation for all clients such as racing or sports games. Given that there has been opportunity to collect information on the level layout beforehand.

V. Methodology

The research question addressed in this project is: how can client side prediction in networked multiplayer first-person shooter games that use dead reckoning be improved? The hypotheses drawn from this question are listed in Table 1.

This project's hypotheses will be confirmed by comparing the projected model from the proposed method against that of a traditional dead reckoned approach. Given that client side prediction is concerned with periods of time of 100ms - 5 seconds (II-A II-B), several samples may be collected from a single participant's session. To compare our proposed method against the established industry standard method of dead reckoning, we look for a small effect size of 0.1. Were we to take 25 samples per session, an a-priori power analysis finds that 30 participants will be required. As outliers can have a negative effect on the accuracy of results from two-way repeated measures tests [53], we will increase this size to 35 participants, with the expectation that some outlying results may need to be discounted to avoid distorting the results. While this paper recognises that a purpose of client side prediction is to simulate what can often be unpredictable behaviour, this project's proposed method solely aims to better simulate predictable behaviour given the level layout, such as slowing down before critical junctures and eliminate impossible behaviour such as wall tunnelling. As acknowledged in section II-F, simulating unpredictable behaviour that extreme outlying results represent is best solved with player modelling, and is beyond the scope of this project.

TABLE I
HYPOTHESES

	Hypothesis	Null Hypothesis	Data Source
1	Using location specific direction trends derived from bot simulations to adjust the dead reckoning error threshold to prompt an increased rate of packet updates at critical junctures will result in a more accurate client side prediction model than a constant error threshold.	Using location specific direction trends derived from bot simulations to adjust the dead reckoning error threshold will not prompt an increased rate of packet updates at critical junctures.	Game Data
2	Using location specific direction trends derived from bot simulations to adjust the dead reckoning error threshold to prompt a decreased rate of packet updates at non critical junctures will negligibly decrease the accuracy of the client side prediction model.	Using location specific direction trends derived from bot simulations to adjust the dead reckoning error threshold to prompt a decreased rate of packet updates at non critical junctures will significantly decrease the accuracy of the client side prediction model.	Game Data
3	Applying location specific acceleration trends to the acceleration factor in dead reckoning algorithms will provide a more accurate projected position at a future time than traditional dead reckoning algorithms.	Applying location specific acceleration trends to the acceleration factor in dead reckoning algorithms will not provide a more accurate projected position at a future time than traditional dead reckoning algorithms.	Game Data

A. Hypotheses

$$C = V_b \cdot D_p \tag{7}$$

1) Confidence-based threshold adjustment

$$\eta_p = \eta_{min} + (\eta_{max} - \eta_{min})|C| \tag{8}$$

2) Trend-based acceleration adjustment

$$A_t = \frac{A_0' + CA_p}{2} \tag{9}$$

$$P_t = P_0 + V_b T_t + \frac{1}{2}A_t T_t^2 \tag{10}$$

Where C is the correlation value between the motion and the trend. D_p is the direction trend at a position P in the level. η_p is the error threshold allowed at position P. η_{min} is the constant minimum threshold allowed by the game. η_{max} is the constant maximum threshold allowed by the game. A_p is the acceleration trend at position P, and A_t is the predicted informed acceleration value for the entity at position P.

(7) Declares the dot product of the blended velocity model found with (3), and the direction trend for the entity's position in the level as a correlation score between the simulation and the data trend.
(8) Uses the absolute value of the correlation amount C as a confidence value, to return a threshold amount for the entity at position P. The returned threshold amount is blended between η_{min} and η_{max} by the $0-1$ confidence value of $|C|$.
(9) Calculates an informed acceleration value A_t as the mean of the last known acceleration value and the acceleration trend for the entity's position in the level, multiplied by the correlation score. i.e. a correlation value of 0 will return half of the last known acceleration.
(10) Takes the simulation projection equation (4) and substitutes A_0' for the informed acceleration model A_t.

Table I shows the hypotheses that are tested in this experiment. Testing these hypotheses requires analysing the results of a standard dead reckoning implementation and the same algorithm amended to include derived level data to alter the algorithm's threshold value, and as an additional acceleration value for the projection of motion. As this requires human participants, ethics approval for this methodology will be sought from Falmouth University's Research Ethics Board.

B. Level design considerations

Participants will be given access to a level in a first person shooter game designed specifically for this project. For simplicity the level will be at a single height throughout. This will allow proof of concept and the ability to visualise the data by use of an overlay as shown in fig 5 without introducing a height vector as shown in fig 7.

To be considered a broadly applicable method, this project must test the hypotheses against a wide range of scenarios. In first-person shooter games this includes wide and thin corridors, obtuse and acute corners, wide open areas, and junctions. As a robust setting for data analysis and sample collection, the level that this project uses must include all of these.

To maintain control over when the prediction events start and end, narrow sections termed as *choke points* will be used to set the participant's controlled entity to a default state suitable for prediction. These will be used to define the angle at which a corner is approached, or from which edge a large room is entered. This must be carefully leveraged against building a typical level that the participant would expect to see in a game. Were the level to resemble a maze, it is reasonable to assume that the participant would move though that level differently to how they would move through a competitive multiplayer environment for which this client side prediction method would ultimately be used. The level and the player then should aesthetically resemble a networked multiplayer first-person shooter environment to encourage the participant to behave within it as such.

The game will be built in Unreal Engine 4.21.0[1] using

---

[1]Unreal Engine 4.21.0: https://www.unrealengine.com/en-US/feed

the Paragon assets[2]. This is due to the project's familiarity with the software, and the professional quality and free use license of the asset pack. The game will be playable on a desktop computer using a mouse and keybord as controls. These controls will mirror the existing standard layout of first-person shooters.

C. Finding trends

This paper's proposed method relies upon effectively and efficiently collecting and storing entity movement data for a level. To gather a set of data that represents possible movement through the level, a large quantity of simulations where AI controlled bots move through the level will be performed and analysed. The quantity of simulations will be restricted by time, although this project aims to simulate 1,000 hours of entity movement by using powerful hardware and increasing the game's update speed. Changes in motion will prompt a recording of an entity's velocity and location replicating the movement data captured by a client side prediction method's state update.

As these bots will adhere to the level's rules regarding possible movement, the resulting data will only represent possible locations and velocities. Using this information, the acceleration values can be derived, as can likely movement dependant on the AI's priorities. This project acknowledges that a number of different AI priorities must be considered.

This data will be then visualised using a heat map overlay to determine and exclude outlying anomalies. While still possible routes through the level, the aim of this work is to identify level specific trends to tend the proposed prediction method towards. Outlying data will distort these trends. From here, the data can be partitioned into critical areas of differing trends for the proposed method to use.

D. Testing the hypotheses

Participants will be told to move between waypoints appearing on the map with the least amount of deaths. These waypoints will be preset to give the project control as to what junctures the player encounters e.g. corners, corridors, and open spaces. Approaching these junctures will prompt the experiment to start a prediction event, from which the results will form a test sample.

Enemies that pose a threat to the player's entity will take the form of fixed place turrets. These will be placed around the junctures to encourage a more cautious play style. The problems that users experience described in III are often a result of client side prediction dead reckoning a future position past a safe area when they are actually still within cover. By encouraging cautious behaviour from participants, we attempt to recreate these problems. To this end, this experiment will not impose a time limit upon the play test.

The number of deaths will not be used directly for statistical analysis as the project is solely concerned with movement data. Instead this will be used to identify and prune outlying

---

participants in regards to engagement and familiarity with first-person shooter controls.

Each entity's health amount will replenish after each juncture. This is essential in maintaining a consistent starting state for each prediction event. Considering the difference in movement between a player controlling a wounded entity and that of an entity with full health falls under player behaviour modelling and outside of the scope of this work.

E. Participants

It can take some time to get used to controlling an entity in a first-person shooter. To avoid having to train participants on how to control an entity in a first person shooter a screening criterion will be employed for potential participants which will require familiarity with first-person shooter controls. Misunderstanding or attempts to deceive this screening will become apparent as an outlying result when considering total deaths in the play test. The question will be: "What is your experience of playing first-person shooter games that use keyboard and mouse controls?" and will require an answer to be given on a slider from the value of "none" to "extremely high". This will feature in the starting sequence for the game and only participants that respond with "none" will not be permitted to continue.

Participants will be asked to agree to a consent form that will be displayed before the game and must be agreed to before the game is played. This will state that statistics about the play session will be recorded, but not which particular statistics to avoid confirmation bias from participants causing unusual movement. Instead, given the context of the game, the apparent goal will be to navigate between the waypoints without dying from enemy entity attacks, replicating how they would move between positions in a competitive multiplayer scenario.

Participants will be randomly sampled from students studying game development related courses at Falmouth University and therefore will fall within the student demographics of age and sex. Sample collection sessions will be performed within The Games Academy building situated upon Falmouth University's campus. As such, all health and safety precautions will be inherited from the faculty's established protocols.

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.

---

[2]Paragon Assets: https://www.unrealengine.com/marketplace/assets?lang=q =Paragon

[7] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Clay-pool, "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. Rangara-jan, "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 Telecom-munication Services*. Springer, 2000, pp. 199–204.

[21] D. Mills, "Network time protocol (version 3) specification, implemen-tation 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. 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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/

[32] (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

[33] 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.

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

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

[37] 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.

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

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

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

[41] (2014) Getting killed around corners. [Online]. Available: https://gamefaqs.gamespot.com/boards/713745-call-of-duty-ghosts/67947149

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

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

[44] 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/

[45] 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

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

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

[48] J. L. Miller and J. Crowcroft, "Avatar movement in world of warcraft battlegrounds," in *Proceedings of the 8th annual workshop on Network and systems support for games*. IEEE Press, 2009, p. 1.

[49] 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.

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

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

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

[53] Two-way repeated measures anova using spss statis-tics. [Online]. Available: https://statistics.laerd.com/spss-tutorials/two-way-repeated-measures-anova-using-spss-statistics.php