Clustering-Based Online Player Modeling

Jason M. Bindewald, Gilbert L. Peterson(✉), and Michael E. Miller

Air Force Institute of Technology, Wright-Patterson AFB, OH, USA
gilbert.peterson@afit.edu

Abstract. Being able to imitate individual players in a game can benefit game development by providing a means to create a variety of autonomous agents and aid understanding of which aspects of game states influence game-play. This paper presents a clustering and locally weighted regression method for modeling and imitating individual players. The algorithm first learns a generic player cluster model that is updated online to capture an individual's game-play tendencies. The models can then be used to play the game or for analysis to identify how different players react to separate aspects of game states. The method is demonstrated on a tablet-based trajectory generation game called *Space Navigator*.

1 Introduction

Automating game-play in a human-like manner is one goal in intelligent gaming research, with applications such as a gaming version of the Turing Test [14] and human-like game avatars [6]. When we move from playing a game generically to playing like a specific individual, the dynamics of the problem change [10]. In complex dynamic environments, it can be difficult to differentiate individual players, because the insights exploited in imitating 'human-like' game-play can become less useful in imitating the idiosyncrasies that differentiate specific individuals' game-play. By learning how to imitate individual player behaviors, we can model more believable opponents [6] and understand what demarcates individual players, which allows a game designer to build robust game personalization [18].

The *Space Navigator* environment provides a test-bed for player modeling in routing tasks, and allows us to see how different game states affect disparate individuals' performance of a routing task. The routing task is a sub-task of several more complex task environments, such as real-time strategy games or air traffic control tasks. Since there is only one action a player needs to take:

M.E. Miller—The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense, or the United States Government. This work was supported in part through the Air Force Office of Scientific Research, Computational Cognition & Robust Decision Making Program (FA9550), James Lawton Program Manager.

The rights of this work are transferred to the extent transferable according to title 17 § 105 U.S.C.

© Springer International Publishing AG (outside the US) 2017
T. Cazenave et al. (Eds.): CGW 2016/GIGA 2016, CCIS 705, pp. 86–100, 2017.
DOI: 10.1007/978-3-319-57969-6_7

draw a trajectory, it is easy for players to understand. However, *Space Navigator*, with its built in dynamism, is complex enough that it is not simple to generate a single 'best input' to any given game state. The dynamism also means that replaying an individual's past play is not possible.

Specifically, we use individual player modeling to enable a trajectory generator that acts in response to game states in a manner that is similar to what a specific individual would have done in the same situation. Individualized response generation enables better automated agents within games that can imitate individual players for reasons such as creating "stand-in" opponents or honing strategy changes by competing against oneself. In addition, the player models can be used by designers to identify where the better players place emphasis and how they play, which can be used to balance gameplay or create meaningful tutorials.

This paper contributes a player modeling paradigm that enables an automated agent to perform response actions in a game that are similar to those that an individual player would have performed. The paradigm is broken into three steps: (1) a cluster-based generic player model is created offline, (2) individual player models hone the generic model online as players interact with the game, and (3) responses to game situations utilize the individual player models to imitate the responses players would have given in similar situations. The resulting player models can point game designers toward the areas of a game state that affect individual behavior in a routing task in more or less significant ways.

The remainder of the paper proceeds as follows. Section 2 reviews related work. Section 3 introduces the *Space Navigator* trajectory routing game. Section 4 presents the online individual player modeling paradigm and the model is then applied to the environment in Sect. 5. Section 6 gives experimental results showing the individual player modeling system's improvements over a generic modeling method for creating trajectories similar to individual users. Section 7 summarizes the findings presented and proposes potential future work.

2 Related Work

Player models can be grouped across four independent facets [15]: domain, purpose, scope, and source. The domain of a player model is either game actions or human reactions. Purpose describes the end for which the player model is implemented: generative player models aim to generate actual data in the environment in place of a human or computer player, while descriptive player models aim to convey information about a player to a human. Scope describes the level of player(s) the model represents: individual (one), class (a group of more than one), universal (all), and hypothetical (other). The source of a player model can be one of four categories: induced - objective measures of actions in a game; interpreted - subjective mappings of actions to a pre-defined category; analytic - theoretical mappings based on the game's design; and synthetic - based on non-measurable influence outside game context. As an example classification, the player model created in [16] for race track generation models individual player tendencies and preferences (Individual), objectively measures actions in the game (Induced), creates

tracks in the actual environment (Generative), and arises from game-play data (Game Action). The player model created here furthers this work by updating the player model online.

One specific area of related work in player modeling involves player decision modeling. Player decision modeling [8] aims to reproduce the decisions that players make in an environment or game. These models don't necessarily care why a given decision was made as long as the decisions can be closely reproduced. Utilizing player decision modeling, procedural personas [7,11] create simple agents that can act as play testers. By capturing the manner in which a given player or set of players makes decisions when faced with specific game states, the personas can help with low-level design decisions.

Past work has used Case-Based Reasoning (CBR) [5] and Learning from Demonstration (LfD) [1] to translate insights gained through player modeling into responses within an environment. The nearest neighbor principle, maintaining that instances of a problem that are a shorter distance apart more closely resemble each other than do instances that are a further distance apart, is used to find relevant past experiences in LfD tasks such as a robot intercepting a ball [1], CBR tasks such as a RoboCup soccer-playing agent [5], or tasks integrating both LfD and CBR such as in real time strategy games [13]. When searching through large databases of past experiences approximate nearest neighbors searches, such as Fast Library for Approximate Nearest Neighbors (FLANN [12]), have proven useful in approximating nearest neighbor searches while maintaining lower order computation times in large search spaces.

3 Application Environment

Space Navigator [2,3] is a tablet computer game similar to *Flight Control* [4] and *Contrails* [9]. Figure 1 shows a screen capture from the game and identifies several key objects within the game. Spaceships appear at set intervals from the screen edges. The player directs each spaceship to its destination planet (designated by similar color) by drawing a line on the game screen using his or her finger. Points accumulate when a ship encounters its destination planet or bonuses that randomly appear throughout the play area. Points decrement when spaceships collide and when a spaceship traverses one of several "no-fly zones" (NFZs) that move throughout the play area at a set time interval. The game ends after five minutes.

4 Methodology

The player modeling paradigm shown in Fig. 2 begins with a cluster-based generic player model created offline (area 1). The generic player model is updated online to adapt to changing player habits and quickly differentiate between players (area 2). Then the online player modeler creates responses to game states that are similar to those that an individual player would have given in response to similar states (area 3).

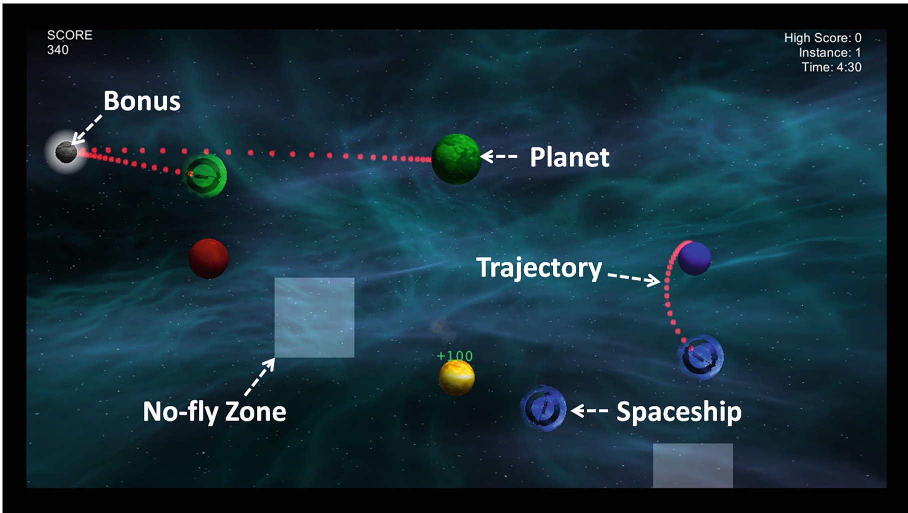


Fig. 1. A *Space Navigator* screen capture highlighting important game objects. (Color figure online)

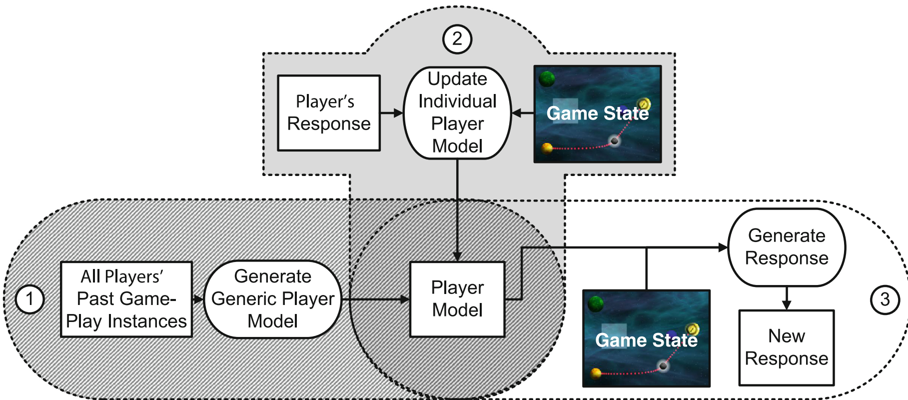


Fig. 2. An online updating individual player modeling paradigm.

State and Response Clustering. Clustering reduces the state-response pairs into a set of representative clusters, dramatically reducing the representation size of a player model. Ward agglomerative clustering [17] provides a baseline for the player modeling method and was proven effective for clustering in trajectory creation game environments in [3,9]. The clustering implemented here takes each game-play instance, containing a state and its associated response, and assigns it to both a state cluster and a response cluster. The number of clusters is a choice left to the practitioner, accounting for the specific environment and resource constraints. A state-response instance mapping from a given state cluster to a given

response cluster demonstrates a proclivity for a player to react with a class of maneuver in a specific type of game situation. By determining the frequency of mappings, common situational responses and outlier actions emerge.

Cluster Outlier Pruning. If a state has only been seen in one instance by one player, that state is unlikely to provide much benefit in predicting future responses. After state and response clustering, clusters with outlier responses are removed first by removing all instances assigned to the least populated response clusters. The cutoff threshold for determining which instances to remove could be either a minimum response cluster size or a percentage of response clusters to remove. For example, due to the distribution of cluster sizes in the *Space Navigator* database we removed instances falling in the bottom 25% of all response clusters according to cluster size (setting a cutoff threshold relies on knowledge of the environment and underlying dataset distribution, and is an area for future work).

Similarly, a response given by only one player in one instance is unlikely to reoccur in future player responses. Outlier state clusters are removed in two ways. First, instances that fall in the bottom 25% of all state clusters according to cluster size are removed, eliminating response clusters that are rare overall. However, removing states not seen by many *different* players is also important. Pruning also removes instances falling into a state cluster encountered by a minimal subset of players, eliminating response clusters reached by an extremely small subset of players.

The resulting player model, $\mathbf{P}_{x,y}$ is the ($x =$ the number of state clusters) $\times$ (y $=$ the number of response clusters) matrix of likelihoods that a given state cluster maps to a given trajectory cluster, such that $\mathbf{p}_{i,j}$ represents the likelihood that state s_i maps to response r_i. This model is created across all game-play instances after cluster pruning is complete. This generic player model, created off-line, forms the baseline for individual player model creation.

4.1 Individual Player Models

For online individual player modeling, the generic player model is updated as an individual plays the game (shaded area 2 of Fig. 2). Over time, the updates shape a player model of an individual player's game-play tendencies. The individual player update trains quickly by weighting learning according to state-response cluster scores.

Algorithm 1 is the online algorithm for learning individual player models. The algorithm begins with the generic player model $\mathbf{P}$. Once a player submits a response in the game environment, the current game state and the response are given as inputs. The algorithm finds the closest state (S_{close}) and response (R_{close}) clusters, and the player model is updated at the intersection of S_{close} and R_{close} by δ_{close}. Then the player model is normalized across all the R values for S_{close} so that the values sum to 1.

There are certain states that provide more information than others. Weighting the increment values for a given state-trajectory pair aids quick learning of

Algorithm 1. Individual player model online update algorithm.

1: **inputs**: $\mathbf{P} = x \times y$ generic player model; $\langle s_{in}, r_{in} \rangle$ = a state-response pair; $\mathbf{M} = \{\langle S_1, R_1 \rangle, \langle S_1, R_2 \rangle, \cdots, \langle S_x, R_y \rangle\}$, all cluster mappings
2: S_{close} = the closest state cluster to state s_{in}
3: $\delta_{close} = q \cdot (\delta_{cp} + \delta_{cmv} + \delta_{pma})$, S_{close}'s update increment weight
4: R_{close} = the closest response cluster to response r_{in}
5: $\mathbf{p}(S_{close}, R_{close}) = \mathbf{p}(S_{close}, R_{close}) + \delta_{close}$
6: **for** $\mathbf{p}(S_{close}, i)$ where $i = 1 \rightarrow y$ **do**
7: $\quad \mathbf{p}(S_{close}, i) = \mathbf{p}(S_{close}, i) / (1 + \delta_{close})$
8: **end for**

player idiosyncrasies. Traits gleaned from the clustered data help determine which state clusters should create larger learning increments, and which states provide minimal information beyond the generic player model. Three traits comprise the update increment, δ. As shown in Algorithm 1, Line 3 these include: cluster population, cluster mapping variance, and previous modeling utility.

Cluster Population: When attempting to learn game-play habits quickly, knowing the expected responses of a player to common game states is important. Weighting δ according to the size of a state cluster in comparison to that of the other state clusters across the entire game-play dataset emphasizes increased learning from common states for an individual player model. States that fall into larger clusters can provide better information for quickly learning to differentiate individual player game-play. To calculate the cluster population trait, all state cluster sizes are calculated and any state cluster with a population above a selected population threshold is given a cluster population trait weight of $\delta_{cp} = 1$ and all other state clusters receive a weight of $\delta_{cp} = 0$.

Cluster Mapping Variance: When mapping state clusters to response clusters, some state clusters will consistently map to a specific response cluster across all players. Other state clusters will consistently map to several response clusters across all players. Very little about a player's game-play tendencies is learned from these two types of state clusters. However, state clusters that map to relatively few clusters per player (intra-player cluster variance), while still varying largely across all players (inter-player cluster variance) can help quickly differentiate players. The state cluster mapping variance ratio is the total number of response clusters to which a state cluster maps across all players divided by the number of response clusters to which the average player maps, essentially the ratio of inter-player cluster variance to the intra-player cluster variance. The cluster mapping variance trait weight, δ_{cmv}, is set according to a cluster variance ratio threshold. All state clusters with a variance ratio above the threshold receive a weight of $\delta_{cmv} = 1$ and all others receive a weight of $\delta_{cmv} = 0$.

Previous Modeling Utility: The last trait involves running Algorithm 1 on the existing game-play data. Running the individual player update model on previous game-play data provides insight into how the model works in the actual game environment. First, Algorithm 1 runs with $\delta = 1$ for all state clusters, training the player model on some subset of a player's game-play data (training set). Then

it iterates through the remaining game-play instances (test set) and generates a response to each presented state, using both the individual player model and the generic player model. This iteration includes each individual player in the game-play dataset. For each test set state, the response most similar to the player's actual response is determined. Each time the individual player model is closer than the generic player model to the actual player response, tally a 'win' for the given state cluster and a 'loss' otherwise. The ratio of wins to losses for each state cluster makes up the previous modeling utility trait. The previous modeling utility trait weight, δ_{pma}, is set according to a previous modeling utility threshold. All state clusters with a previous modeling utility above the threshold receive a weight of $\delta_{pma} = 1$ and all others receive a weight of $\delta_{pma} = 0$.

Calculating δ: When Algorithm 1 runs, δ is set to the sum of all trait weights for the given state cluster multiplied by some value q which is an experimental update increment set by the player. Line 3 shows how δ is calculated as a sum of the previously discussed trait weights.

4.2 Generate Response

Since response generation is environment specific, this section demonstrates the response generation section shown in area 3 of Fig. 2 for a trajectory generation task. The resulting trajectory generator creates trajectories that imitate a specific player's game-play, using the cluster weights in $\mathbf{P}$ from either a generic or learned player model.

The trajectory response generation algorithm takes as input: the number of trajectories to weight and combine for each response (k), the number of state and trajectory clusters (x and y respectively), the re-sampled trajectory size (μ), a new state (s_{new}), a player model ($\mathbf{P}$), and the set of all state-trajectory cluster mappings ($\mathbf{M}$). Line 2 begins by creating an empty trajectory of length μ which will hold the trajectory generator's response to s_{new}. Line 3 then finds the state cluster (S_{close}) to which s_{new} maps. $\mathbf{P}_{close}$, created in Line 4, contains a set of likelihoods. $\mathbf{P}_{close}$ holds the likelihoods of the k most likely trajectory clusters to which state cluster S_{close} maps.

The loop at Line 5 then builds the trajectory response to s_{new}. Humans tend to think in terms of 'full maneuvers' when generating trajectories–specifically for very quick trajectory generation tasks such as trajectory creation games [9]–rather than creating trajectories one point at a time. Therefore, the *Space Navigator* trajectory response generator creates full maneuver trajectories. Line 7 finds the instance assigned to both state cluster S_{close} and trajectory cluster T_i with the state closest to s_{new}. The response to this state is then weighted according to the likelihoods in $\mathbf{P}$. The loop in Line 9, then combines the k trajectories using a weighted average for each of the μ points of the trajectory. The weighted average trajectory points are normalized across the k weights used for the trajectory combination in Line 13 and returned as the response to state s_{new} according to the player model $\mathbf{P}$.

Algorithm 2. Trajectory response generation algorithm.

1: **inputs**: k = the number of trajectories to combine; x = the number of state clusters; y = the number of trajectory clusters; μ = the re-sampled trajectory size; s_{new} = a state we have not seen before; $\mathbf{P}$ = an $x \times y$ player model; $\mathbf{M} = \{\langle S_1, T_1 \rangle, \langle S_1, T_2 \rangle, \cdots, \langle S_x, T_y \rangle\}$, all state-trajectory cluster mappings
2: **initialize**: $t_{new}(\mu) \leftarrow$ an empty trajectory of μ points
3: S_{close} = the closest state cluster to state s_{new}
4: $\mathbf{P}_{close} = \max_k \left[\mathbf{P}_{S_{close},(z|\forall z \in 1,\ldots,y)} \right]$
5: **for** each $P_{close,i} \in \mathbf{P}_{close}$ **do**
6: T_i = the trajectory cluster associated with $P_{close,i}$
7: $s_{close,i} \leftarrow$ state closest to s_{new} in $\langle S_{close}, T_i \rangle$
8: $t_{close,i} \leftarrow$ the response trajectory to $s_{close,i}$
9: **for** $\nu = 1 \rightarrow \mu$ **do**
10: $t_{new}(\nu) = t_{new}(\nu) + t_{close,i}(\nu) \cdot P_{close,i}$
11: **end for**
12: **end for**
13: **return** $t_{new} = t_{new} / \sum_{i=1}^{k} P_{close,i}$

5 Environment Considerations

This section demonstrates how the player modeling paradigm can be applied to generating trajectory responses in *Space Navigator*. First, an initial data capture experiment is outlined. Then, solutions are presented to two environment specific challenges: developing a state representation and comparing disparate trajectories.

5.1 Initial Data Capture Experiment

An initial experiment captured a corpus of game-play data for further comparison and benchmarking of human game-play [3]. Data was collected from 32 participants playing 16 five-minute instances of *Space Navigator*. The instances represented four difficulty combinations, with two specific settings changing: (1) the number of NFZs and (2) the rate at which new ships appear. The environment captures data associated with the game state when the player draws a trajectory, including: time stamp, current score, ship spawn rate, NFZ move rate, bonus spawn interval, bonus info (number, location, and lifespan of each), NFZ info (number, location, and lifespan of each), other ship info (number, ship ID number, location, orientation, trajectory points, and lifespan of each), destination planet location, selected ship info (current ship's location, ship ID number, orientation, lifespan, and time to draw the trajectory), and selected ship's trajectory points. The final collected dataset consists of 63,030 instances.

5.2 State Representation

Space Navigator states are dynamic both in number and location of objects. The resulting infinite number of configurations makes individual state identification

difficult. To reduce feature vector size, the state representation contains only the elements of a state that directly affect a player's score (other ships, bonuses, and NFZs) scaled to a uniform size, along with a feature indicating the relative length of the spaceship's original distance from its destination. Algorithm 3 describes the state-space feature vector creation process.

Algorithm 3. State-space feature vector creation algorithm.

1: **input**: L = the straight-line trajectory from spaceship to destination planet.
2: **initialize**: $\eta \in [0.0 \cdots 1.0)$ = weighting variable; s = empty array (length 19); $zoneCount = 1$
3: Translate all objects equally s.t. the selected spaceship is located at the origin.
4: Rotate all objects in state-space s.t. L lies along the X-axis.
5: Scale state-space s.t. L lies along the line segment from $(0, 0)$ to $(1, 0)$.
6: **for** each object type $\vartheta \in (OtherShip, Bonus, NFZ)$ **do**
7:     **for** each zone $z = 1 \rightarrow 6$ **do**
8:         $zoneCount = zoneCount + 1$
9:         **for** each object o of type ϑ in zone z **do**
10:             d_o = the shortest distance of o from L
11:             $w_o = e^{-(\eta \cdot d_o)^2}$ ▷ Gaussian weight function
12:             $s[zoneCount] = s[zoneCount] + w_o$
13:         **end for**
14:     **end for**
15: **end for**
16: $s[19]$ = the non-transformed straight-line trajectory length
17: **return** s, normalized between $[0, 1]$

The algorithm first transforms the state-space features against a straight-line trajectory frame in Line 1. Lines 3–5 transform the state-space along the straight-line trajectory such that disparate trajectories can be compared in the state-space. The loop at Line 6 accounts for different element types and the loop at Line 7 divides the state-space into six zones as shown in Fig. 3. This effectively divides the state-space into left and right regions, each with three zones with relation to the spaceship's straight-line path: behind the spaceship, along the path, and beyond the destination.

To compare disparate numbers of objects, the loop beginning in Line 9 uses a weighting method similar to that used in [5], collecting a weight score (s) for each object within the zone. This weight score is calculated using a Gaussian weighting function based on the minimum distance an object is from the straight-line trajectory. Figure 3 shows the transformation of the state into a feature vector using Algorithm 3. The state-space is transformed in relation to the straight-line trajectory, and a value is assigned to each "entity type + zone" pair accordingly. For example, Zone 1 has a bonus value of 0.11 and other ship and NFZ values of 0.00, since it only contains one bonus. Lastly, the straight-line trajectory distance is captured. This accounts for the different tactics used when ships are at different distances from their destination. The resulting state representation values are normalized between zero and one.

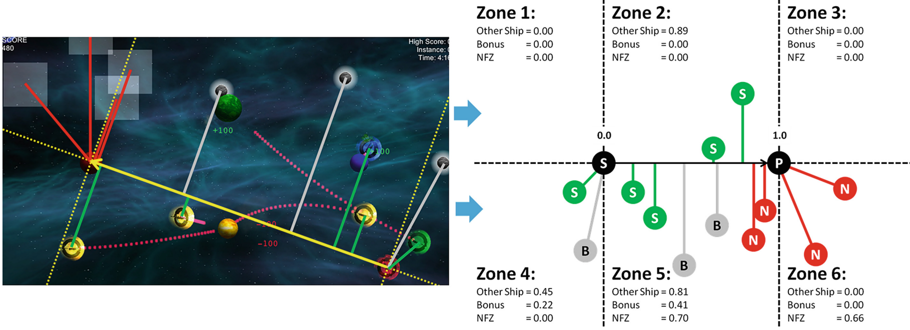


Fig. 3. The six zones surrounding the straight line trajectory in a *Space Navigator* state representation and the state representation calculated with Algorithm 3.

5.3 Trajectory Comparison

Trajectory generation requires a method to compare disparate trajectories. Trajectory re-sampling addresses the fact that trajectories generated within *Space Navigator* vary in composition, containing differing numbers of points and point locations. Re-sampling begins by keeping the same start and end points, and iterates until the re-sampled trajectory is filled. The process first finds the proportional relative position (p_m) of a point. The proportional relative position indicates where the i-th point would have fallen in the original trajectory and may fall somewhere between two points. The proportional distance (d_m) that p_m falls from the previous point in the old trajectory (p_0) is the relative distance that the i-th re-sampled point falls from the previous point. To compare trajectories, the target number of points is set to 50 (approximately the mean trajectory length in the initial data capture) for re-sampling all the trajectories.

Re-sampling has two advantages: the re-sampling process remains the same for both trajectories that are too long and too short and maintains the distribution of points along the trajectory. A long or short distance between two consecutive points, remains in the re-sampled trajectory. This ensures that trajectories drawn quickly or slowly maintain those sampling characteristics despite the fact that the draw rate influences the number of points in the trajectory. Since feature vector creation geometrically transforms a state, the trajectories generated in response to the state are transformed in the same manner, ensuring the state-space and trajectory response are positioned in the same state space.

To ensure the trajectories generated in *Space Navigator* are similar to those of an individual player, a distance measure captures the objective elements of trajectory similarity. The Euclidean trajectory distance treats every trajectory of i (x, y) points, as a $2i$-dimensional point in Euclidean space where each x and each y value in the trajectory represents a dimension. The distance between two trajectories is the simple Euclidean distance between the two $2i$-dimensional point representations of the trajectories. A 35-participant human-subject study confirmed that Euclidean trajectory distance not only distinguished between

trajectories computationally, but also according to human conceptions of trajectory similarity.

6 Experiment and Results

This section describes testing of the online individual player modeling trajectory generator and presents insights gained from the experiment. The results show that, with a limited amount of training data, the individual player modeling trajectory generator is able to create trajectories more similar to those of a given player than a generic player-modeling trajectory generator. Additionally, the results show the model provides insights for a better understanding of what separates different players' game-play via comparison to the generic player model.

6.1 Experiment Settings

The experiment compares trajectories created with the generic player model, the individual player model, and a generator that always draws a straight line between the spaceship and its destination planet. The first five games are set aside as a training dataset and the next eleven games as a testing dataset. Five training games (equivalent to 25 min of play) was chosen as a benchmark for learning an individual player model to force the system to quickly pull insights that would manifest in later game-play. For each of 32 players, the individual player model is trained on the five-game training dataset using Algorithm 1 with the trait score weights. Next, each state in the given player's testing set is presented to all three trajectory generators and the difference between the generated and actual trajectories recorded. Experimental values for the individual player model are set as follows: update increment $(q) = 0.01$, cluster population threshold = 240, cluster mapping variance threshold = 17.0, and previous modeling utility threshold = 3.0.

The three learning thresholds specific to *Space Navigator* are: (1) state cluster population threshold = 240 (set at a value of one standard deviation over the mean cluster size), forty of 500 state clusters received a cluster population weight of $\delta_{cp} = 1$; (2) cluster mapping variance ratio threshold = 17, with 461 of 500 state clusters receiving a cluster variance weight of $\delta_{cmv} = 1$; and (3) previous modeling utility threshold = 3, with 442 of 500 clusters receiving a previous modeling utility score of $\delta_{pma} = 1$.

To account for the indistinguishability of shorter trajectories, results were removed for state-trajectory pairs with straight-line trajectory length less than approximately 3.5 cm on tablets with 29.5 cm screens used for experiments. This distance was chosen as it represents the trajectory length at which an accuracy one standard deviation below the mean was reached.

6.2 Individual Player Modeling Results

Testing of the game-play databases shows that the trajectories generated using the individual player model significantly improved individual player imitation results

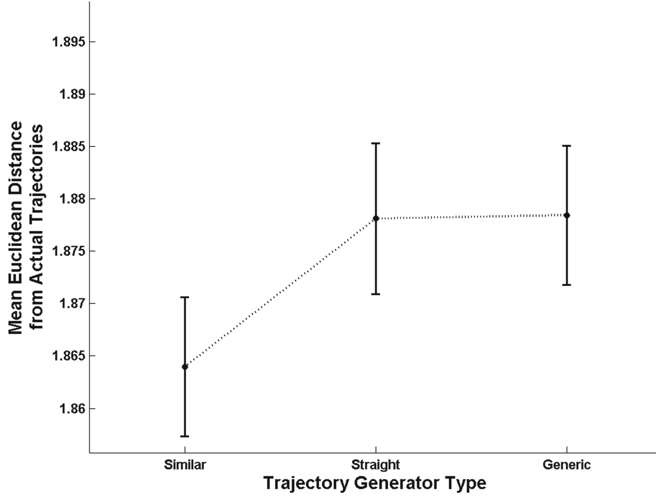


Fig. 4. Euclidean trajectory distance between generated trajectories and actual trajectory responses.

when compared to those generated by the generic player model and the straight line trajectory generator. Table 1 and Fig. 4 show results comparing trajectories generated using each database with the actual trajectory provided by the player, showing the mean Euclidean trajectory distance and standard error of the mean across all 32 players and instances.

Table 1. Mean and standard error of the Euclidean trajectory distances (in *SpaceNavigator* environment meters).

Database	Mean dist	Std err
Individual player model	1.8640	±0.0063
Straight line generator	1.8781	±0.0069
Generic player model	1.8784	±0.0063

The individual player model generator provides an improvement over the other models. The mean Euclidean trajectory distance of 1.8640 provides a statistically significant improvement over the straight line and generic player models, as standard error across all instances from all 32 players does not overlap with the latter two player models. The similar player model improves the generic databases accuracy by learning more from a selected subset of presented states to ensure that the player model more accurately generates similar trajectories.

6.3 Individual Player Model Insight Generation

The changes in player model learning value for each element of a state representation show which aspects of the state influence game-play. This enables a better understanding of what distinguishes individual game-play within the game environment.

Table 2 shows the results of a Pearson's linear correlation between the mean learning value change of each state cluster across all 32 players and the state representation values of the associated state cluster centroids. The results show that there is a statistically significant negative correlation between the mean learning value changes and all of the zones, but some changes are much larger than others. The overall negative correlation arises among object/zone pairs intuitively: high object/zone pair scores imply a large or close presence of a given object type, constraining the possible trajectories. There is more differentiability of player actions when more freedom of trajectory movement is available.

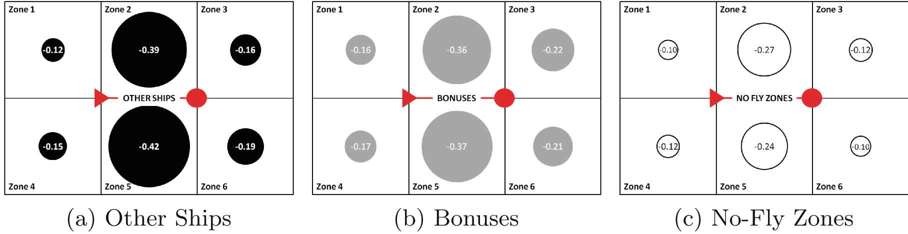
Table 2. Correlation of each state representation value with the mean change in associated state cluster learning values in player models

Zone	Pearson's r	p-value	Pearson's r	p-value	Pearson's r	p-value
	Other Ships		*Bonuses*		*NFZs*	
1	−0.1227	0.0060	−0.1569	0.0004	−0.1002	0.0251
2	−0.3911	0.0000	−0.3552	0.0000	−0.2749	0.0000
3	−0.1616	0.0003	−0.2212	0.0000	−0.1184	0.0080
4	−0.1465	0.0010	−0.1662	0.0002	−0.1159	0.0095
5	−0.4244	0.0000	−0.3693	0.0000	−0.2398	0.0000
6	−0.1903	0.0000	−0.2056	0.0000	−0.1040	0.0200
Dist	−0.6434	0.0000				

With the ship-to-planet distance feature, longer distances correlate to smaller learning value changes among player models, with the strongest correlation of all features: r of -0.6434 and p-value <0.0001. Possible explanations for this behavior include: (1) players are more constrained over long distances, (2) as distances get longer, the variance in the way an individual player draws trajectories in similar situations increases, (3) shorter distances capture consistent tendencies that carry along to distinguish individual game-play over time.

Another aspect that Table 2 shows is the importance of the middle zones in comparison to the 'before' and 'after' zones. Figure 5 illustrates this point graphically. The r values show that the middle two zones provide a larger influence on the amount of change in the learning values. For example, in Fig. 5a the r values for zones two and five are more than double those of any other zone. This idea is somewhat intuitive as this is the area that the ship will traverse, providing the most likely cause for interaction with objects of any given type.

The results also provide insight into the relative value that players place on certain types of objects. For example, determining the correlation coefficients of different Object/Zone Pairs can show that No Fly Zones in the middle two zones provide a significantly smaller influence on learning value changes than other ships do in the same zones.



(a) Other Ships (b) Bonuses (c) No-Fly Zones

Fig. 5. Graphical representation of the correlation coefficient for each Object Type/Zone score with the mean change in learning values in player models.

Three examples of how player modeling insights can be used in game applications involve training, game design, and player automation. Player models can be used to find places where specific users who are doing really well properly value certain actions over others. Proper valuations can then be communicated to players during training within the environment. Another example is that, we can use the player modeling insights to design point structures to more closely align with the way players perceive the value of different object types. Lastly, modeling a specific player enables the designer to incorporate an automated player to play like a specific expert or current user within the game.

7 Conclusions and Future Work

The online individual player modeling paradigm presented in this paper is able to generate trajectories similar to those of a specific *Space Navigator* player. The system is able to operate online without needing to perform time-consuming offline calculations to update individual player models. Additionally, the gains in individual player imitation are found in a relatively small number of games (five games, totaling 25 min). The player models developed to imitate players also allow for a better understanding of what traits of a given state provide understanding of player differences which occur for different states.

This work provides opportunities for several areas of future work. Further studies will research the effects of using the trajectory generator to act as an automated aid for players interacting with the *Space Navigator* game. Additionally, further analysis of the player modeling methods could yield further insights into how much differentiation of individual players can be gained over different amounts of time. Moreover, imitating individual players could provide helpful insights in determining how experts play *Space Navigator* to aid in experiments to learn how to improve player training.

References

1. Argall, B., Browning, B., Veloso, M.: Learning by demonstration with critique from a human teacher. In: Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction, pp. 57–64. ACM (2007)
2. Bindewald, J.M., Miller, M.E., Peterson, G.L.: A function-to-task process model for adaptive automation system design. Int. J. Hum. Comput. Stud. **72**(12), 822–834 (2014)
3. Bindewald, J.M., Peterson, G.L., Miller, M.E.: Trajectory generation with player modeling. In: Barbosa, D., Milios, E. (eds.) CANADIAN AI 2015. LNCS (LNAI), vol. 9091, pp. 42–49. Springer, Cham (2015). doi:10.1007/978-3-319-18356-5_4
4. Firemint Party Ltd.: Flight control, December 2011. https://itunes.apple.com/us/app/flight-control/id306220440?mt=8
5. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating robocup players. In: FLAIRS Conference, pp. 251–256 (2008)
6. Gamez, D., Fountas, Z., Fidjeland, A.K.: A neurally controlled computer game avatar with human like behavior. IEEE Trans. Comput. Intell. AI Games **5**(1), 1–14 (2013)
7. Holmgård, C., Liapis, A., Togelius, J., Yannakakis, G.N.: Evolving personas for player decision modeling. In: 2014 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2014)
8. Holmgård, C., Liapis, A., Togelius, J., Yannakakis, G.N.: Generative agents for player decision modeling in games. Foundations of Digital Games (2014)
9. Huang, V., Huang, H., Thatipamala, S., Tomlin, C.J.: Contrails: crowd-sourced learning of human models in an aircraft landing game. In: Proceedings of the AIAA GNC Conference (2013)
10. Kemmerling, M., Ackermann, N., Preuss, M.: Making Diplomacy bots individual. In: Hingston, P. (ed.) Believable Bots, pp. 265–288. Springer, Heidelberg (2012)
11. Liapis, A., Holmgård, C., Yannakakis, G.N., Togelius, J.: Procedural personas as critics for dungeon generation. In: Mora, A.M., Squillero, G. (eds.) EvoApplications 2015. LNCS, vol. 9028, pp. 331–343. Springer, Cham (2015). doi:10.1007/978-3-319-16549-3_27
12. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1), 2 (2009)
13. Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in StarCraft. Comput. Intell. AI Games **5**(4), 293–311 (2013)
14. Schrum, J., Karpov, I.V., Miikkulainen, R.: Human-like combat behaviour via multiobjective neuroevolution. In: Hingston, P. (ed.) Believable Bots, pp. 119–150. Springer, Heidelberg (2012)
15. Smith, A.M., Lewis, C., Hullet, K., Sullivan, A.: An inclusive view of player modeling. In: The 6th International Conference on Foundations of Digital Games, pp. 301–303. ACM (2011)
16. Togelius, J., De Nardi, R., Lucas, S.M.: Making racing fun through player modeling and track evolution. In: Proceedings Optimizing Player Satisfaction in Computer and Physical Games, p. 61 (2006)
17. Ward, J.H.: Hierarchical grouping to optimize an objective function. J. Am. Stat. Assoc. **58**(301), 236–244 (1963)
18. Yu, H., Riedl, M.O.: Personalized interactive narratives via sequential recommendation of plot points. IEEE Trans. Comput. Intell. AI Games **6**(2), 174–187 (2014)