Table of Contents

Chapter 1

Modelling Openings

In this chapter we introduce the problem of identifying opening lines of play in high-fidelity combat simulations. Our approach is to model the movement of individual combatants, and we use **discrete trajectories** – ordered sequences of points in space – as a natural representation for this movement. We will provide a notation and a variety of operations with which to represent and manipulate such trajectory data.

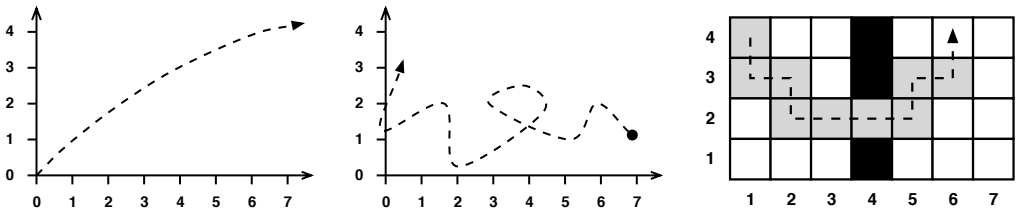
An opening book for a game like Chess is useful because it enables us to understand opening plays. That is, if a particular opening line of play is described by our opening book, we should be able to **recognize and predict** an opening when we see it. We can furthermore make judgements about how well a particular opening is working for us or for a general population of players; i.e., given experience, we can **evaluate** between different **categories** of openings. We may also be able to make generalisations about a type of player, i.e., **classify** a player, based on the openings s/he executes; in this chapter, we will show that an important question is whether or not human players are distinguishable from human players in a particular combat simulation.

1.1 Trajectories

In this section we introduce and formally define **discrete trajectories**, the core unit of analysis in this study. Trajectories are commonly used to describe moving points, but more generally can represent changes in a vector variable over time. For instance, a trajectory can be used to describe changes in the location of a ball as it rolls down a hill, changes in an

athlete's heart-rate during periods of exercise and rest, a sequence of board states in a game of Chess, or the spatial location of a character moving around in a combat simulation.

1.1.1 Defining Trajectories



(a) $y = x^{0.8}$ (b) A loop in $\mathbb{R}^2$ (c) A grid-world trajectory

Figure 1.1: Examples of trajectories

Though trajectories are easier to grasp intuitively using diagrams such as those in Figure 1.1, we need a formal definition and a notation with which to represent them. The formal definition is precise, and a consistent notation will help us to describe algorithms that manipulate trajectories.

Definition 1.1.1 (Discrete Trajectory). A discrete trajectory X is defined by a time-indexed set $\{X_t : t \in T\}$, where:

- $X_t \in \mathbb{R}^n$ represents a point in n-dimensional space occurring at time index t, and
- $T \subset \mathbb{N}^+$ is a set of time indices.

For a discrete trajectory $X = \{X_t : t \in T\}$, we refer to the **length** of trajectory X as the size of the set that defines it, i.e., $|X|$. If there is a constant difference between the adjacent indices in T, X is called a **constant-interval trajectory**.[1] If, however, the differences between adjacent indices in T are variable, then T is called a **variable-interval trajectory**.[2] Finally, since trajectories are defined by a set of time-indexed points, we naturally use $\emptyset$ denote the **empty trajectory**.

[1] Automated systems that monitor an easily observable process such as the state of a computer video game [KB06], often produce constant-interval trajectories.

[2] It is sometimes difficult to take point measurements at constant intervals, for example in studies that involve personal interviews or require mobile objects of variable speed to pass a waypoint.

Example 1.1.1. The following are examples of discrete trajectories:

- $X = \{(10, 10)_1, (20, 30)_3, (30, 50)_4\}$,
- $Y = \{(1, 4, 0)_1, (3, 2, 0)_5, (5, 3, 0)_8, (6, 4, 0)_{10}\}$,
- $Z = \{(56, 67)_1, (60, 70)_2, (65, 78)_3, (69, 86)_4\} \longleftarrow$ (a constant-interval trajectory).

When only part of a trajectory is relevant, it can be desirable to isolate a subsequence of points from the full trajectory. Specifically, we define the **sub-trajectory** of a trajectory X from α to β to be a trajectory containing the subsequence of the points in X whose time indices are delimited between α and β.

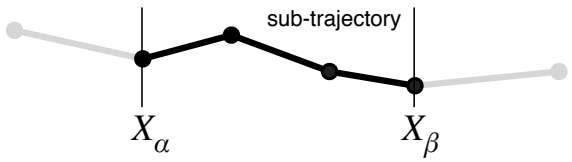


Figure 1.2: Illustration of a sub-trajectory.

Definition 1.1.2 (Sub-trajectory). Let $X = \{X_t : t \in T\}$ be a trajectory. A sub-trajectory of X from α to β, where $0 < \alpha \leq \beta \leq |X|$, is a trajectory of length $(\alpha - \beta + 1)$ and is defined as follows.

$$X\Big|_\alpha^\beta = \{X_t \in X, \alpha \leq t \leq \beta\}$$

This notation can be used to specify a sub-trajectory containing a single point, but not a trajectory containing no points.

Example 1.1.2. The following are examples of sub-trajectories.

- Let $X = \{(10, 10)_2, (20, 30)_3, (30, 50)_4\}$,
  - The sub-trajectory from 2 to 3 is $X\Big|_2^3 = \{(10, 10)_2, (20, 30)_3\}$,
  - The sub-trajectory from 3 to 4 is $X\Big|_3^4 = \{(20, 30)_3, (30, 50)_4\}$.
- Let $Y = \{(1, 4, 0)_1, (3, 2, 0)_3, (5, 3, 0)_8, (6, 4, 0)_1 0\}$,
  - The sub-trajectory from 1 to 5 is $Y\Big|_1^5 = \{(1, 4, 0)_1, (3, 2, 0)_3\}$,
  - The sub-trajectory at 8 is $Y\Big|_8^8 = Y_8 = \{(5, 3, 0)_8\}$.

1.1.2 Similarity

Similarity is important in any classification or pattern recognition task, as well as in cluster analysis in which objects are grouped into related clusters. Several techniques exist to determine the similarity (or, equivalently, the distance) between two trajectories. One possibility is to measure the distance between the partitioned line segments of trajectories [LHW07] but the necessary pre-processing risks removing important points from a trajectory.[3] An alternative is to measure distance by smoothly interpolating between points [YiAS03] at the risk of introducing inaccurate points to a trajectory.[4] Taking the Euclidean (or 2-norm) distance between the vectors of points within two trajectories avoids these risks. Direct quote: "It is well-known, paramaterless, trivial to implement, and predates data mining by several decades." [KK02] and a number of sophisticated competitors have been shown inferior on one-dimensional (time series) trajectory data [KK02].

Definition 1.1.3 (Euclidean Trajectory Distance). Let X and Y be discrete trajectories, $X = \{X_t : t \in T\}$ and $Y = \{Y_t : t \in S\}$. The **Euclidean trajectory** distance between X and Y is:

$$\mathrm{dist}(X, Y) = \left(\sum_{i \in T \cap S} |X_i - Y_i|^2 \right)^{1/2}. \tag{1.1}$$

We use the following notation to indicate the distance between two sub-trajectories:

$$\mathrm{dist}(X, Y)\Big|_{\alpha}^{\beta} = \mathrm{dist}\left(X\Big|_{\alpha}^{\beta}, Y\Big|_{\alpha}^{\beta} \right) \tag{1.2}$$

1.1.3 Trajectory Generators

- Points along a traj can describe many phenomena.
- Something causes, or generates them. We will call this cause a trajectory generator.
  - A traj generator explicitly describes ways of creating trajectories.

[3] Pre-processing to remove points along which a trajectory does not change rapidly, according to the information theoretic *minimum description length* principle, may help to distill important line-segment information [LHW07], but subtle changes along a trajectory are not always unimportant.

[4] Trajectories recorded at discrete time intervals risk omitting local zigzags [CJB+02] that correspond to important terrain (strategically, physically, or otherwise), which makes interpolation dangerous.

  - It implicitly describes a continuous, multi-variate probability distribution over a state-space of trajectories.

- Without a description of the generator, we must rely on analysing samples from the distribution it implies.

- A particular traj generator may be prone to generating similar trajectories.

Definition 1.1.4 (Trajectory Generator). A trajectory generator is an autonomous or human-controlled process that produces a trajectory over time. It may be stochastic (its future state represented by some probability distribution) or deterministic. They create a time series. Has an internal state, i.e., its output is conditionally dependent on the history of its previous output.

The following are a few examples of trajectory generators.

1. A mathematical function, e.g., $\vec{x}(t_i) = i^2$.

2. A computer program, e.g. an agent learning its environment generates trajectories that can be useful in understanding the nature of its environment [MB01].

3. The result of a human or computer program interfacing with a combat simulation to make strategic gains.

4. Decisions made that change board state in a game of chess.

5. Real-world phenomena where the generating process is unknown or not well understood, e.g., the paths of hurricanes and the migration patterns of tagged wildlife [LHW07].

By far the most relevant trajectory generator in this study is the movement of computer and human-controlled characters in combat simulations.

1.1.4 Operations on Trajectories

- Eventual aim is to design algorithms that manipulate trajs.

- Our data is constant-interval discrete trajs. This means we can treat them as vectors [GS99].

- We describe basic arithmetic operations:

  - Addition,

- subtraction, and
- scalar multiplication

- We describe concatenation

Definition 1.1.5 (Trajectory Addition, Subtraction, and Scalar Multiplication). Let A and B be discrete trajectories, with identical time intervals $\vec{\delta} \in \mathbb{R}^m$ between points and with m points, i.e., $X = \{X_t : t \in T\}$ and $Y = \{Y_t : t \in S\}$. The addition, subtraction, and scalar multiplication of trajectories are defined as follows.

- **Addition**: $X + Y = \{Z_t : Z_t = X_t + Y_t, t \in T \cap S\}$
- **Subtraction**: $X - Y = \{Z_t : Z_t = X_t - Y_t, t \in T \cap S\}$
- **Scalar Multiplication**: $c \cdot X = \{Z_t : Z_t = c \cdot X_t, t \in T\}$

Example 1.1.3.

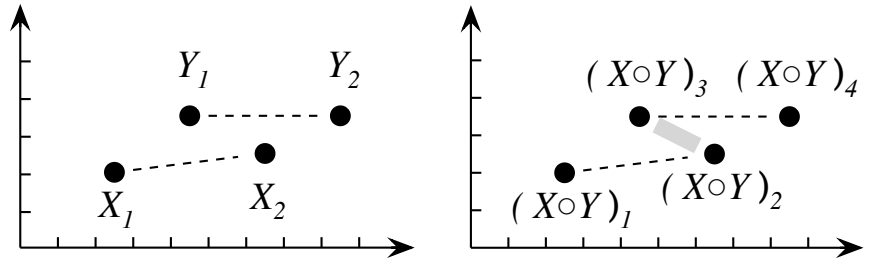


Figure 1.3: The concatenation of two trajectories in $\mathbb{R}^2$.

Definition 1.1.6 (Concatenation). Let X and Y be two discrete trajectories, $X = \{X_t : t \in T\}$ and $Y = \{Y_t : t \in S\}$. We have:

- **Trajectory concatenation** of X with another trajectory, i.e. The concatenation of X and Y is defined by time-shifting the elements of Y forward so that they occur after the elements in X, i.e., by adding $m = \max(T)$ to each time index in Y, i.e.,

$$X \circ Y = X \cup \{Z_i : Z_{i+m} = Y_i, i \in S\}$$

- **Concatenative identity**: Concatenation of X with the empty trajectory, Finally, the **empty trajectory** $\emptyset$ is defined as the concatenative identity for discrete trajectories. That is, for any trajectory X,

$$X \circ \emptyset = \emptyset \circ X = X.$$

Example 1.1.4.

1.1.5 An Example Algorithm

In the following example we describe an algorithm that tracks two trajectory generators f and g at regular time intervals of length Δt. The algorithm stores the average point value generated by f and g and stores it in a discrete trajectory A.

Algorithm 1.1.1 Example Algorithm

1: $A \leftarrow \emptyset$
2: $t \leftarrow 1$
3: **while** recording **do**
4: $\quad F \leftarrow 0.5 \cdot \{F_1 : F_1 = f(t)\}$
5: $\quad G \leftarrow 0.5 \cdot \{G_1 : G_1 = g(t)\}$
6: $\quad A \leftarrow A \circ (F + G)$
7: $\quad t \leftarrow t + 1$
8: **end while**
9: **return** A

Algorithm 1.1.1 begins with the initialisation of A to the empty trajectory, $\emptyset$ (line 1), and the current time-step t to 0 (line 2). Once inside the main loop, we define a trajectory with a single point, $f(t)$, *multiply* it (Definition 1.1.5) by the scalar 0.5, and store the resulting trajectory in F (line 3). We repeat the process for $g(t)$ and G (line 4). Next, we *add* (Definition 1.1.5) F and G and *concatenate* (Definition 1.1.6) A with the result (line 5). Completing an iteration of the loop, we increment t (line 6). Figure 1.4 illustrates a possible execution of Algorithm 1.1.1 for $f : \mathbb{R} \longrightarrow \mathbb{R}$, $g : \mathbb{R} \longrightarrow \mathbb{R}$, and $\Delta t = 1$.

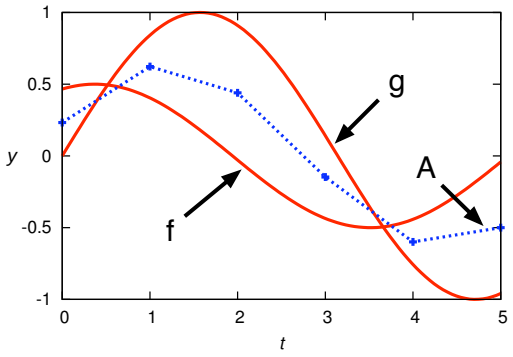


Figure 1.4: Algorithm 1.1.1.

1.2 Problem Definition

This thesis focuses on problems defined by the tuple $(\mathbb{T}, \mathbb{O})$: $\mathbb{T}$ is a set of discrete trajectories, and $\mathbb{O}$ is a set of categorical outcomes (e.g., wins and losses). Each $\mathbb{T}_i \in \mathbb{T}$ is the result of a combatant moving around in a combat environment that is familiar to that combatant, and results in the corresponding game outcome $\mathbb{O}_{\sqsupset} \mathbb{T}$. With respect to this tuple, we intend to answer the following questions:

- Is human behaviour measurably different from the behaviour of bots that were designed to emulate humans? (**classification**)
- Are there groups of highly similar trajectories in $\mathbb{T}$ that correspond to predictable behaviour? (**prediction**)
- Are game outcomes statistically dependent on trajectories? (**correlation**)

Trajectory generators in combat simulations – i.e., the players – explicily describe a process, and implicitly define a continuous multivariate probability distribution. We wish to model these generators. In high fidelity environments and situations in which the underlying processes are not completely understood or easily represented, we rely on approximate models. Modelling is important!

Recall that openings provide topics for analysis (i.e., activities that can be named/talked about).

We define $\mathbb{T}$ to be a set of observed trajectories, and $\mathbb{O}$ to be a set of outcomes, where each $\mathbb{O}_i \in \mathbb{O}$ maps to the end-game outcome of $\mathbb{T}_i \in \mathbb{T}$. Thus, we look at problems of the form $(\mathbb{T}, \mathbb{O})$. In particular:

- A variety of sophisticated agents designed to emulate humans [Boo04] already come pre-packaged with a variety of combat simulations [Gam] [Val] [Stu]. Does it matter whether $\mathbb{T}$ comes from autonomous computer-controlled agents or humans? Why not just use the representation they use, i.e., waypoints?
- Can models of the multivariate probability distribution followed by $\mathbb{T}$ be used to recognize and then **predict**/forecast an opening line of play?
- Can the same models assist us in making judgements about how well a particular opening will do, i.e., **evaluating** that opening?
- An analogy: the asteroid and the approximate modelling of Newtonian mechanics.

- Nature is the trajectory generator for an asteroid; a human is the trajectory generator for a character.
- Its trajectory is the path it takes through space, relative to some fixed point of observation; a character's trajectory is the path it follows in a simulated combat environment.
- It is modelled by Newtonian mechanics, an approximate description of the underlying mechanisms that cause the asteroid to move; how to model a character in a way that provides an explanation of the underlying mechanisms?

1.2.1 Trajectory Prediction

- Answering: do categories of $\mathbb{T}$ exist?
- forecast future movement by recognizing opening trajectories before they are complete, and
- Consistent processes can be accurately modelled.
- Random processes cannot.
- Looking for consistency in $\mathbb{T}$.
- Behaviour in known combat sims can be consistent [Man01].
- Example: paper rock scissors. The more consistent your opponent is, the more you can benefit from modelling him.
- Task is to predict future points along a trajectory from its initial segment.
- Benefits include better opposition and complementary behaviour,
- and being able to move scripted and autonomous behaviour [Lid00] towards something that is more human.
- I.e., if we can accurately predict, then we can act.

Task Specification

Figure 1.5 accompanies the following specification of the description task. Let A be a trajectory of length m. The **initial segment** of A is a sub-trajectory (Definition 1.1.2) from

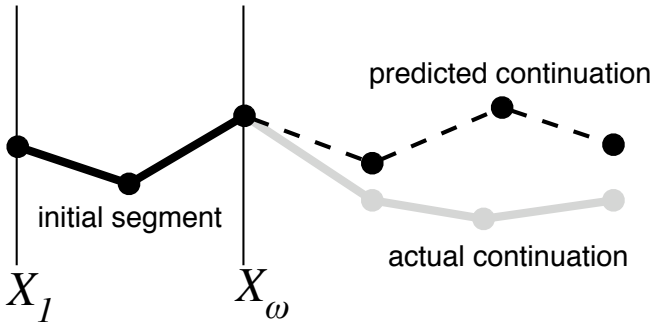


Figure 1.5: A sketch of the trajectory prediction problem. Left, a trajectory's initial segment is shown, its points occurring between time indices α and ω. The trajectory's *actual* continuation after ω is shown in grey, and a *predicted* continuation, is shown with dashed lines. The predicted continuation is scored in terms of its distance from the actual continuation.

the initial point in A to an intermediate point $0 < \omega < m$, i.e.,

$$A\Big|_1^\omega = \{A_t \in A : t \leq \omega\}$$

The **actual continuation** of A from this initial segment is the remaining sub-trajectory, i.e.,

$$A\Big|_{\omega+1}^{\max(T)} = \{A_t \in A : t > \omega\}$$

A model is tasked with recovering a **predicted continuation** C that is as close as possible to the actual continuation of A, i.e.,

$$C = \{C_t : t \in T > \omega\} \approx A\Big|_{\omega+1}^{\max(T)}.$$

In particular, given a set of example trajectories D_{train}, and a set of test trajectories D_{test}, a trajectory model is submitted to the prediction task as follows.

1. **Training Phase**: A trajectory model M is constructed using each trajectory $T \in D_{\text{train}}$.[5]

2. **Testing Phase**: Given the initial segment of each $A \in D_{\text{test}}$, M is used to create C,

[5]The construction of specific models is covered in detail in Chapter **??**.

a *predicted continuation* of A.

We measure a model M's performance at this task with the Euclidean distance (Definition 1.1.3) between the actual subsequent measurements of $A \in D_{\text{test}}$ and the trajectory model's predicted continuation:

$$\text{err} = \text{dist}\left(C, A\Big|_{\omega+1}^{\max(T)}\right).$$

1.2.2 Turing Test

Finally, this thesis also sets out to answer the question of whether human data is necessary for such an analysis. When designers build bots [Boo04], they create a representation of the environment that could be useful for describing opening play. But how similar are the activities conducted by bots to those concuted by humans? We shall see.

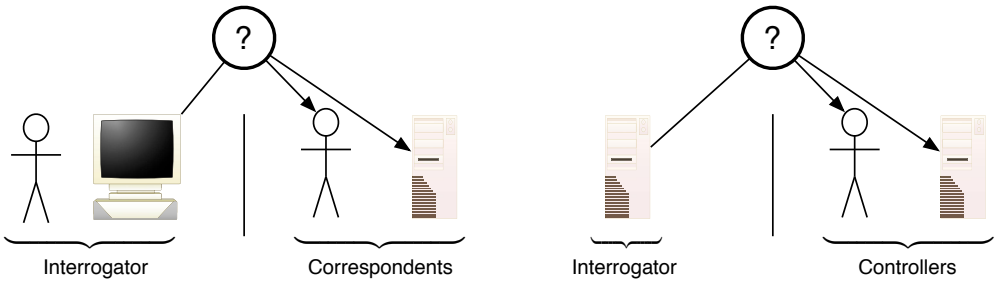
- Answers: is it necessary to have humans? Or can we just deploy bots [Dar07]. Is there anything particular about human gameplay data, or do bots that are designed to behave like humans, with the same sensory input [Boo04] do the same thing?
- i.e. is a model derived from HUMAN gameplay data detectably different from one derived from BOT gameplay data?
- are bots 'good enough?'
- If categorising $\mathbb{T}$ produces a model that performs poorly at classification, then either the two groups have the same categories OR the categories fasil to capture the nuances of individual playing styles.
- If on the other hand categorisation produces a model that excels at classification, then the two groups have different categories AND the categories succeed in capturing the individual playing styles.

We propose that a model that explains strategic behaviour should be able to assist in discerning between the underlying trajectory generators (Definition **??**). In particular, we propose that a model should be able to discern between *computer* controllers and *human* controllers.

This task bears some resemblance to the Turing Test [Tur50], originally proposed as a measure of machine intelligence, and more recently to measure how convincingly computer controllers imitate humans in combat simulations [LD00] [Liv06]. In the original

Turing test, a human interrogator sits at a plain-text terminal (Figure 1.6(a)). The terminal is connected on the other end to a correspondent, which is either a human with an identical terminal or a machine designed to imitate a human correspondent, i.e. to converse as if it were itself a human. The interrogator is tasked with determining whether the correspondent is a human or a machine; a machine that convinces the interrogator that it is human is said to have *passed* the Turing test.

In our proposed classification task, the interrogator is replaced with another computer, the correspondents become trajectory generators (either human or computer controllers), and the domain changes. Additionally, the controllers receive no feedback from the interrogator, and instead act of their own volition. Thus, instead of *conversing* with a controller, the interrogator merely observes a trajectory that the controller generates, and uses its experience with other controllers to classify the underlying trajectory generator (Figure 1.6(b)).



(a) The original Turing Test

(b) Alternative classification

Figure 1.6: Experimental set-up for the Turing test, left, and for the classification task we identify as being important for a model that explains behaviour in combat simulations, right.

Task Specification

Given a set of example trajectories D_{human} which are generated by human controllers and a set of example trajectories D_{bot} which are generated by computer controllers, and a set of unlabelled test trajectories D_{test}, a trajectory model is submitted to the classification task as follows.

1. **Training Phase**: Two trajectory models are constructed, one to explain the underlying controller of the *computer* generated trajectories in D_{bot} and one to explain the

underlying controller of the *human* generated trajectories in D_{human}.

2. **Testing Phase**: For each trajectory $A \in D_{\text{test}}$ the models are used to classify/recognize the underlying controller, and a label is assigned.

We measure a model's explanatory ability at this task in terms of the mean frequency with which it correctly labels test instances $B \in D_{\text{test}}$.

1.2.3 Categorisation

Do categories exist? Natural language descriptions of strategies on the web suggest the answer is yes. Intuition suggests yes. Manninen suggests yes [Man01].

One of the most important thing that a model of opening gameplay can do is to identify topics for analysis in $\mathbf{T}$, identify categories of play.

We propose that a model that explains strategic behaviour should be able to assist in determining the *value*, or *expected reward* for following a particular trajectory. In other words, the payoff analysis task looks at a trajectory model's suitability for making quantitative comparisons between different trajectories.

There is strong motivation for having a model with this ability. For *human* players, combat simulations are meant to be won; being able to quantitatively compare two types of strategic behaviour is useful for improving performance and identifying a team's weaknesses via post-game analysis. For A.I. controllers, the goal is to provide fun competition for human players, and to put on a good show when dumb things happen [Lid03]. Unfortunately, existing techniques involve tuning coarsely-grained difficulty levels (such as *easy*, *medium*, and *hard*), or by manipulating a few toggles tuned to control a bot's behaviour [LD00]. Alternatively, behaviour can be altered over rounds based on learning [BSP05].

Task Specification

Definition 1.2.1 (Game Outcome). An integer $o \in \mathbb{R}$ which is similar to a reward indicating a real-valued payoff for a particular outcome for the team whose outcome correponds to that value.

Primarily, it is important that a model be able to evaluate trajectories on a relative scale, for instance in terms of the *probability* with which certain outcomes in the combat simulation occur (Figure **??**). For example, if following a particular trajectory is likely to result in

a win 50% of the time, then a model will be able to associate a trajectory with that probable outcome.

Additionally, we require a means by which to measure the statistical significance of a model's payoff evaluation. A model may be able to determine the sample probability of a particular outcome, but if the significance is weak, i.e., if the model has not been exposed to an adequate sample of trajectories, then we cannot be confident in its evaluation. Thus, a model should be able to be used to generate a contingency table, which maps categories of strategies to the significance between different trajectories (Figure **??**). This representation is helpful because a variety of techniques exist to determine the statistical independence of categorical data represented in this form.

1.3 Summary

In this chapter we introduced the problem of identifying opening lines of play in high-fidelity combat situations, naming the important criteria of a model that should be able to predict, classify, and categorise openings. Along the way, we introduced trajectories, the core unit of analysis in this study, as data structures with which to represent the movements of combatants; we additionally introduced a number of operations with which to manipulate these trajectories. In the next chapter we introduce our approach to the problem, as well as two baselines against which to measure our approach's ability to predict and classify the trajectories generated by a combatant.

Bibliography

[Boo04] Michael Booth. The Official Counter-Strike Bot. In *Proceedings of the Game Developers' Conference*, 2004.

[BSP05] Sander Bakkes, Pieter Spronck, and Eric Postma. Best-Response Learning of Team Behaviour in Quake III. In *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

[CJB+02] B. Chandrasekaran, J. R. Josephson, B. Banerjee, U. Kurup, and R. Winkler. "diagrammatic reasoning in support of situation understanding and planning". In *Proceedings of the 23rd Army Science Conference*, FL, 2002.

[Dar07] Christian Darken. Level Annotation and Test by Autonomous Exploration. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE)*, Stanford, California, 2007.

[Gam] Epic Games. *The Unreal Universe*. http://www.unreal.com/ (7 Mar. 2008).

[GS99] Scott Gaffney and Padhraic Smyth. Trajectory Clustering with Mixtures of Regression Models. In *Knowledge Discovery and Data Mining*, pages 63–72, 1999.

[KB06] Jason Keir and Michael Barlow. Data-Capture for 1-st Person Simulations: Design, Implementation, and Benefits. In *Proceedings of SimTecT 2006 Conference*, Melbourne, Australia, 2006.

[KK02] Eamonn Keogh and Shruti Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *The 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[LD00] John E. Laird and John C. Duchi. Creating Human-like Synthetic Characters with Multiple skill levels: A Case Study using the Soar Quakebot. In *AAAI 2000 Fall Symposium Series: Simulating Human Agents*, November 2000.

[LHW07] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory Clustering: A Partition-and-Group Framework. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 593–604, Beijing, China, June 2007.

[Lid00] Lars Liden. The Integration of Scripted and Autonomous Behaviors Through Task Management. In *Artificial Intelligence and Interactive Entertainment: Papers from the 2001 AAAI Symposium*, pages 51–55, 2000.

[Lid03] Lars Liden. Artificial Stupidity: The Art of Intentional Mistakes. In *AI Game Programming Wisdom II*, 2003.

[Liv06] Daniel Livingstone. Turing's Test and Believable AI in Games. *Comput. Entertain.*, 4(1):6, 2006.

[Man01] Tony Manninen. Virtual Team Interactions in Networked Multimedia Games - Case: "Counter-Strike" - Multi-player 3D Action Game. In *Proceedings of PRESENCE2001 Conference*, Philadelphia, Pennsylvania, 2001.

[MB01] Amy McGovern and Andrew G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the 18th International Conference on Machine Learning*, pages 361–368, San Francisco, California, 2001. Morgan Kaufmann.

[Stu] Bungie Studios. *The Halo Series*. http://halo.bungie.org/ (18 Mar. 2008).

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):443–460, October 1950.

[Val] Valve. *Counter-Strike: Source*. http://www.counter-strike.net/ (7 Mar. 2008).

[YiAS03] Yutaka Yanagisawa, Jun ichi Akahani, and Tetsuji Satoh. Shape-Based Similarity Query for Trajectory of Mobile Objects. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 63–77, London, UK, 2003. Springer-Verlag.