University of Alberta

Library Release Form

Name of Author: David Christopher Ferguson Rayner

Title of Thesis: Analysing Openings in Tactical Simulations

Degree: Master of Science

Year this Degree Granted: 2008

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

David Christopher Ferguson Rayner
Department of Computing Science

Date: _____

University of Alberta

ANALYSING OPENINGS IN TACTICAL SIMULATIONS

by

David Christopher Ferguson Rayner

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2008

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Analysing Openings in Tactical Simulations** submitted by David Christopher Ferguson Rayner in partial fulfillment of the requirements for the degree of **Master of Science**.

_____

Vadim Bulitko
Supervisor

_____

Michael Bowling

_____

Marcia Spetch
External Examiner

Date: _____

Abstract

‘Openings’ are strategies for playing the beginning of a game, and are often described as sequences of low-level actions (as in chess, for example). This makes it easy to query a database of recorded games for useful information, such as an opening’s success rate, popularity, and more.

‘Tactical shooters’ are video games featuring pseudo-realistic tactical simulations. Here, openings are conventionally described using natural language, which is general enough to capture an opening’s individual variations but provides no formal means of recognising it.

Using Valve’s *Counter-Strike: Source* as a research platform, this thesis describes the use of cluster analysis over a database of recorded games to formally distinguish between openings in tactical simulations. Empirical results reveal significant correlations between openings and game outcomes. The same process is also shown to effectively model player behaviour, and has potential future application to player classification and the development of believable computer-controlled players.

Acknowledgements

I am grateful to:

My parents and family for pointing me forward;

My supervisor, Vadim Bulitko, for his insightful suggestions and patience beyond belief. He has simultaneously given me guidance and the freedom to pursue topics I found interesting;

Michael Bowling and Marcia Spetch for serving on my defense committee, for their careful reading, and for their insightful feedback;

The members of the IRCL and CSAI research groups at the University of Alberta, for sharing their ideas with me and listening to mine;

Countless friends and colleagues, some of whom provided direct feedback on this dissertation, many of whom I bounced ideas off of, and all of whom kept me sane.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Openings are sequences of planned actions taken at the start of a game. Chess openings, for example, have been studied for hundreds of years and are an important part of strategic play. The *Stonewall Attack* is one such opening (Figure 1.1); it represents only a small subset of the all board states at ply-5 (i.e., after White moves 5 times). Using algebraic notation, one way to describe the Stonewall Attack is:

$$\text{1.d4, 2.e3, 3.f4 4.c3 5.Bd3.} \tag{1.1}$$

Such a description leaves no ambiguity as to whether the Stonewall Attack has been executed in a game or not. And since the notation (1.1) can be easily parsed by a computer program, the opening's rate of success, use by notable players, and popularity with time can easily be tracked in a computer database—this benefits players and historians alike.

Tactical shooters are a genre of computer game that takes place in computerised combat simulations. To promote realism, these games typically define an enormous number of states and accept a broad range of input. Consequently, as in real life, players can perform what is effectively the same 'opening' in many different ways. This freedom of action precludes the use of a conventional notation that can easily be parsed by computers.

This thesis presents an analysis of recorded human gameplay data for the purpose of automatically discovering common openings in tactical shooters. The approach I investigate involves grouping highly similar openings by means of cluster analysis, and using these groupings to define *prototype* openings over which all other openings are assumed to be individual variations. This process is shown to reveal meaningful statistically significant correlations between the openings a player enacts and the outcome of a game.

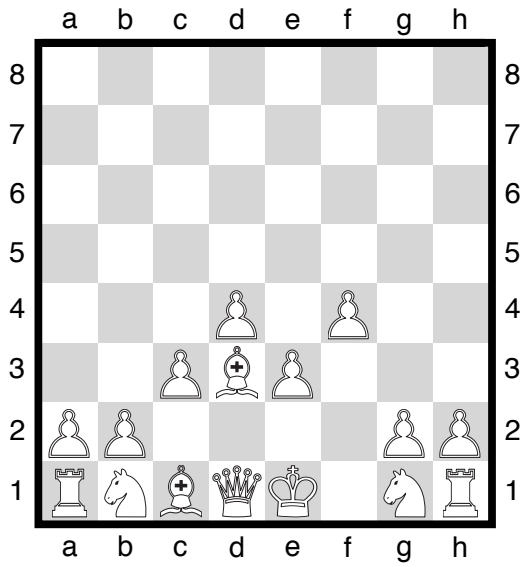


Figure 1.1: The Stonewall Attack is an opening in chess. It is characterised by staggered pawns and a well-positioned bishop. Because the board is small and chess has simple rules, chess openings are easy to describe as an assignment of pieces to positions.

1.1 The Role of Openings

Opening theory, the body of knowledge surrounding how to best play openings, is engaged in a complementary relationship with artificial intelligence research. Well-specified openings can be employed by a computer program that lacks the ability to focus its search of a massive problem space, thereby saving time and programmer effort. Conversely, computers can be programmed to develop new openings, or even complete theoretical solutions to games [Allis, 1988; Tesauro, 1995; Yang *et al.*, 2001; Schaeffer *et al.*, 2007]. As well, computer databases of games can be categorised and annotated automatically (i.e., using relatively simple algorithms) which can be useful in uncovering good openings to play.

Openings also serve as a unit of analysis for analysing gameplay. The study of a game's openings can reveal how common tactics have evolved over time; for instance, while there exist millions of well-defined chess openings, modern players are highly specialised and employ a mere fraction of them in practice [Charness, 1991]. A game's openings can also be used to describe the preferences of individual players. For example, knowledge of an opponent's preferred openings can be used to create a stronger opposition.

Figure 1.2: Top-right, Bohemia Interactive's *Operation Flashpoint* has received attention in academia for its potential application to combat training. Bottom-left, Destineer's *Close Combat: First to Fight* was developed with input from trained marines to improve realism.

1.2 Tactical Shooters

Tactical shooters are a popular genre of computer video game in which teams of independent entities compete in violent, pseudo-realistic virtual environments. Usually the entities are human-controlled, but computer-controlled entities (or **bots**) can participate as well.

Culturally important (i.e., both immensely popular [Nielsen Media Research, 2007] and the subject of ethnographical research [Vesterby, 2002]), tactical shooters juxtapose entertainment and serious combat training. Commercial releases such as Bohemia Interactive's *Operation Flashpoint* [2001a] and Destineer's *Close Combat* [2005a] (Figure 1.2)[1] show promise in simulating aspects of real-life combat training [Wray *et al.*, 2004; Lewis and Barlow, 2005], while input from trained combat personnel has gone into developing more realistic commercial computer games [Tamte, 2004; Reike and Boon, 2008].

[1]The image in Figure 1.2, top-right, is cited after Bohemia Interactive Studio [2001b], and the image in Figure 1.2, bottom-left, is cited after Destineer [2005b];

Figure 1.3: The first-person perspective in Valve's *Counter-Strike: Source* [2004] provides additional information to players via a *radar* (top-left) that indicates nearby combatants.

A defining feature of the genre is the way in which players interact with the game world. Each player controls one of several independent entities (or 'combatants'), which reacts precisely to input from sensitive control devices. Rather than monitor their combatant's precise location, heading and momentum, players see the game world through a graphical first-person perspective (Figure 1.3);[2] aspects of the game world that are occluded by obstacles or outside the field of view are unknown to the player. To provide a believable simulation, the game world is updated rapidly,[3] and good timing and precise reflex control are vital. The number of openings defined by such a process can exceed 2^{22}.[4]

Beneath their graphical veneer, tactical shooters have many strategic elements in common with classic strategy games. Individual combatants resemble game pieces; they begin play in predefined positions, they vie for positional development, they can be used to capture (i.e., remove from play) other combatants, and they can themselves be captured. Conceptually, the opening lines of play tend to be even simpler than in classic strategy games.

[2]The image in Figure 1.3 is cited after Valve [2008].

[3]One source recommends a minimum of 66 updates per second [Invision Gaming, 2007]

[4]See Appendix B for additional details.

1.3 Openings in Tactical Shooters

Natural language is flexible and widespread, and captures all of the individual variations of a particular opening that can occur in simulation space. Player communities typically use natural language to describe openings. For example, Aaltonen [2005] contributes the following description of opening play for Valve's *Counter-Strike: Source*:[5]

> *The longer route is ... right and through the set of double doors. The shorter route runs parallel to the middle, between the bases. [Site]* B *is one of the smallest and easiest to protect ... while [site]* A*'s openness is more difficult.*

Unfortunately, natural language can be ambiguous, and can make incorrect assumptions about a reader's experience. Indeed, descriptions such as the one above risk misinterpretation, and can potentially leave out details that are important to effectively carrying an opening out. Worst of all, supporting evidence for the execution of any one such opening is necessarily anecdotal because there is no algorithmic way to recognise it in play.

A naive alternative is to describe openings as precise sequences of timed, low-level actions. But players have neither the precise control nor the ability to monitor the low-level variables involved in enacting these openings. Moreover, this approach would treat as separate the thousands of variations over what many players would agree is the same opening.

1.4 Discretising Opening Space

The preceding section describes how natural language can be interpreted by human players, if with some ambiguity, but not by automated systems. The simple alternative of using low-level actions makes openings that are difficult for human players to act out and are too specific to embody general gameplay knowledge. This implies a need for a new way to represent the openings in tactical shooters—in particular, a representation that accommodates imperfect human reflex control while retaining a formal structure. This motivates *discretising* the **opening space** (Definition 1.4.1) of a game in such a way that each discretised region contains all of the variations of a specific class of opening.

[5] For a graphical overview of the environment being described in this example, see Figure 6.2 on page 48.

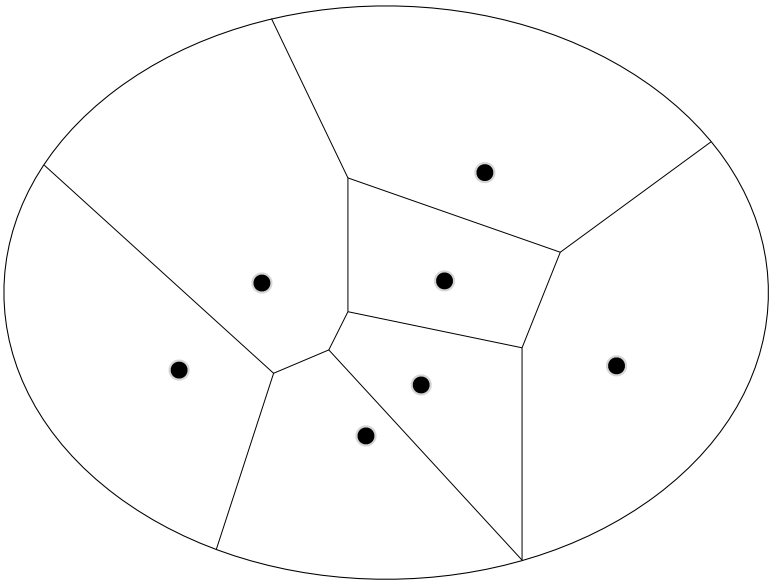


Figure 1.4: A Voronoi tessellation discretises a metric space into unique regions as defined by a set of "generating points" (indicated here with circles). A point in space is included in a region if it is closer to that region's generating point than any other generating point.

Definition 1.4.1 (**Opening Space**)**.** The opening space of a game contains all of the initial sequences of actions that can possibly be executed over a fixed time-span.

For the purpose of analogy, suppose it were possible to map an opening space onto a two-dimensional plane. Further, suppose that this projection retained the relative distance between openings. The problem of discretising this space into a set of regions, each containing all individual variations of an opening, is akin to drawing lines on the plane to distinguish the separate openings.

This thesis explores the use of cluster analysis as a heuristic tool for drawing these lines, albeit in a multi-dimensional space. In particular, the clustering will have the effect of dividing spatially related openings into groups. From each group, a representative point can be identified, which can serve as a generating point for a Voronoi tessellation of the space of all possible openings (Figure 1.4 shows an example in two-dimensional space).

1.5 Summary

This chapter introduced openings as planned sequences of actions taken at the start of a game. I first illustrated that documented openings can have several benefits to the players of many games: well-defined openings facilitate knowledge transfer and define a set of features which can be used to describe individual play. Then I illustrated that tactical shooters are poised to benefit from precisely defined openings, but present several challenges. These include: (1) the lack of an unambiguous notation with which to describe play, (2) the hiding of many important state variables from players, and (3) the difficulty for players in trying to produce 'exact' action sequences. These observations motivate selectively discretising the space of all possible openings into well-defined regions, so as to provide a consistent notation with which to document the openings in these complex domains.

Chapter 2

Related Work

This chapter presents a survey of related work from a variety of fields. First I survey computationally-driven tactical behaviour in games (i.e., methodologies and implemented systems for generating realistic tactical behaviour). Next I survey work in believability testing, the measurement of how life-like a computer-controlled character (or agent) is. Next, although research into computing game openings has largely focussed on classic games, several developments in this field bear mentioning. Finally, a treatment of research into modelling the movements of goal-directed agents provides context for my research.

2.1 Programming Tactical Behaviour

The programming of realistic computerised tactical characters can be subdivided by methodology. **Preprogramming** involves procedurally defining behaviour as reactions to preconditions; **planning** involves declaratively defining behaviour as states, actions, and goals; and **adapting** agents employ machine learning to develop superior performance.

2.1.1 Preprogramming

Preprogramming,[1] the process by which behaviour is defined as *reactions* to *conditions*, may also be thought of as a process of *annotation* [Doyle, 1999]. When an agent comes upon an annotation (i.e., when that annotation's preconditions are met), the agent reactively executes a pre-specified set of instructions. A simple example of preprogramming might be an agent that drinks from a cup of water until it is empty, then refills it at the sink.

[1] Preprogramming is also often referred to as *scripting* because the execution of preprogrammed instructions resembles a stage actor's recitation of a written script.

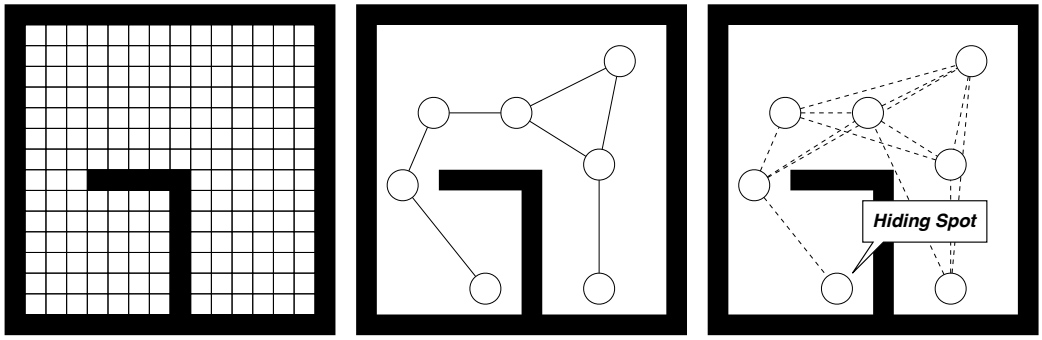


Figure 2.1: Left, the state-space in a video game can be very large. Centre, a waypoint graph can reduce the state-space and ease the computational burden of having an agent decide upon its next action. Right, the waypoint graph can be annotated with *visibility information* which, for example, may be useful in identifying hiding spots in an environment.

A popular type of annotation is a **waypoint graph** (Figure 2.1), which labels locations (or waypoints) with information about what actions may be performed there [Lidén, 2000].[2] Recent literature on computerised combat often takes advantage of the popularity of waypoint graphs. Lidén [2001; 2002] proposes automatically annotating waypoints with information about which other waypoints there are lines of sight to, showing that this can reveal the tactical uses of an environment. Because an untested waypoint graph might define paths and lines of sight that are, in practice, inaccessible by an agent, Darken [2007] proposes tasking the agents themselves with automatically exploring and annotating an environment.

Preprogramming can specify various other agent behaviours. Khoo and Zubek [2002] use potential fields to guide agent motion; Barlow and Morrison [2005] preprogram realistic battlefield stressors such as the effects of suppression fire and injury; and Lidén [2003] describes ways in which agents can be preprogrammed react to their own tactical errors.

Annotations can be arbitrarily complex and interrelated, and can cause seemingly intelligent behaviour, but they are fragile in dynamic environments. If the aforementioned cup-filling agent is placed in a world where cups can be stolen and sinks can begin to dispense pink ooze, it must be redesigned with reactions to these new situations. Preprogramming may also yield inexpert behaviour. Manninen [2001] observes that human expert players learn all of the tactical "advantage points" in an environment; this expertise may come to rival the expertise of the designer originally tasked with preprogramming an agent.

[2] Waypoint graphs are also used for *planning* paths to new locations—planning is covered in Section 2.1.2.

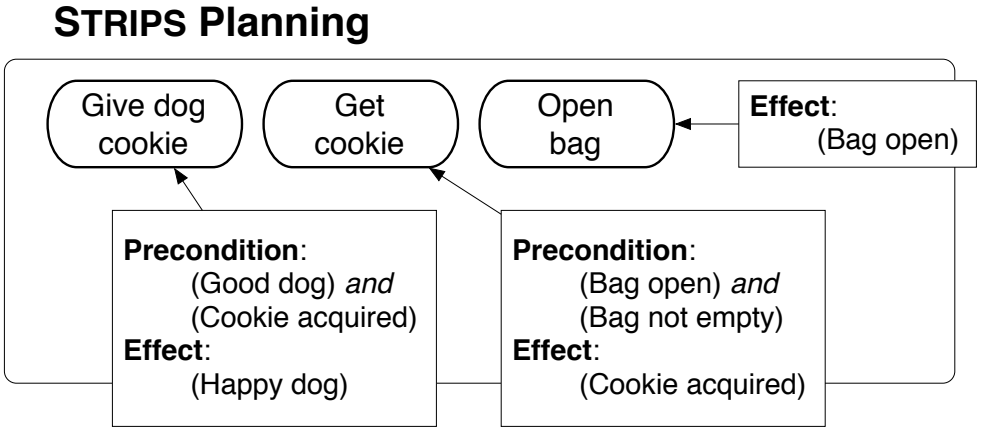


Figure 2.2: In STRIPS planning, low-level actions bear no explicit relationship to one-another. Imagine a STRIPS planner tasked with bringing the world from the state (*Good dog*) to a state specifying (*Happy dog*). It is up to the planner to determine in which order the illustrated actions should be executed so that (1) the preconditions of each successive action are met, and (2) the eventual world state includes the literal (*Happy dog*).

2.1.2 Planning

By specifying states, actions, and goals, designers can declaratively define a wide range of behaviours. A simple example of planning is a *Dog Training* scenario: to reinforce good behaviour, a trainer needs to alter the world state so that a good dog becomes a happy dog. With sufficient world knowledge, a trainer can plan an action sequence to affect this change. Various planning systems have been used to specify the behaviour of a variety of tactical agents. These include systems that plan on the level of primitive actions, systems that plan according to a hierarchical decomposition of a complex task, and hybrid approaches.

In STRIPS-style[3] planning [Fikes and Nilsson, 1971] agents reason on the level of primitive actions to determine how to bring the current world state to a goal world state (Figure 2.2). Orkin [2006] uses a variation of STRIPS-style planning called GOAP[4] to achieve realistic and critically acclaimed tactical behaviour in Monolith Productions' *F.E.A.R.* [2005]. The enemy agents in *F.E.A.R.* have a *killEnemy* goal which can be accomplished with the *attackFromCover* action; a prerequisite of the *attackFromCover* action is that the agent takes cover, which can be met by executing a *gotoNode* action.

[3] STRIPS stands for STanford Research Institute Problem Solver.

[4] GOAP stands for Goal-Oriented Action Planning. GOAP uses a heuristic search to formulate plans and accommodates other computational requirements met in developing real-time games [Orkin, 2004].

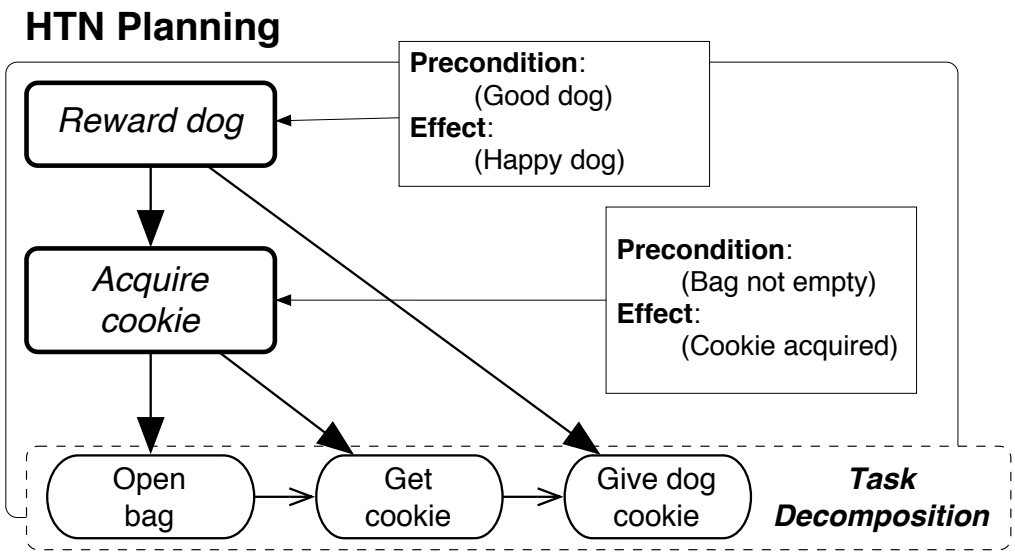


Figure 2.3: In HTN planning, low-level actions are hierarchically composed by a designer into abstract operators. For instance, the *acquire cookie* abstract operator represents two successive low-level actions.

Hierarchical task network planning, or "HTN planning," refers to a planning approach wherein tasks are hierarchically composed of simpler tasks (Figure 2.3). An overview of HTN planning is presented in [Erol *et al.*, 1994]. Hoang *et al.* [2005] successfully use HTN planning to coordinate a team of agents in a tactical environment where the objective is to capture and defend pre-set waypoints. This method of control is shown to outperform approaches based on simple finite-state machines.

The SOAR cognitive architecture [Laird *et al.*, 1987] incorporates a decision cycle that resembles HTN planning: SOAR can specify *abstract operators* that are defined by a hierarchical composition of increasingly concrete operators. However, the SOAR architecture additionally can define *new* abstract operators whenever necessary in a process called *chunking*. Laird [2001] contributes the SOAR-based *Quakebot* agent, which is shown to benefit from chunking when anticipating its opponents in id Software's *Quake II* [1997]. In particular, the Quakebot develops a variety of *new* anticipative measures that it caches for quicker response times. SOAR has also been used as the underlying controlling mechanism for Wray *et al.*'s MOUTBot [2004], an agent designed to embody some of the essential behavioural traits of characters in military training simulations.

Planning systems enable designers to implicitly specify an enormous range of behaviours. However, defining appropriate actions and goals in highly interactive and dynamic settings can be time consuming, especially as requirements change. An alternative is to have agents discover for themselves which goals and actions are most appropriate, adapting through autonomous exploration or by observing and imitating an ideal model of behaviour.

2.1.3 Adapting

A variety of machine learning techniques have been employed in an attempt to improve upon or automatically learn new tactical behaviour. One growing body of research investigates **imitation learning** wherein an agent is given data generated by some idealised *model* agent, typically a human, and tasked with generating similar behaviours. Geisler [2002] investigates imitation learning of some rudimentary aspects of combat control, including agent movement, heading, and acceleration in Raven Software's *Soldier of Fortune II* [2002]. Similarly, Zanetti and Rhalibi [2004] use neural networks to perform imitation learning of an agent's movement and aiming behaviour in id Software's *Quake III* [1999].

Other research focusses on adapting to develop superior tactics; Cole *et al.* [2004] use genetic algorithms to tune an agent's weapon selection and 'aggression' to attain better performance in Valve's *Counter-Strike* [2000]. Darken *et al.* [2004] use agent-centred sensor grids to guide agents to nearby cover; Paull and Darken [2004] extend this work by having the sensor grid additionally discover annotated waypoints in an environment.

An additional body of work concerns the adaptive organisation of a *team* of agents. Existing research is less concerned with the massive state-spaces team organisation might imply. Instead, there is a focus on learning abstract behavioural parameters based samples taken from highly stochastic processes (these challenges are elucidated by Spronck's RAGI[5] framework [2005]). Bakkes *et al.*'s TEAM-2 [2005], an extension of Bakkes *et al.*'s earlier work [2004], learns to assign roles to agents on a team as conditioned on a general representation of the world state. Smith *et al.*'s RETALIATE[6] [2007] employs undiscounted, on-policy Q-learning [Watkins, 1989] to adaptively learn which waypoints to direct agents towards, conditioned on a world state defined by activity at those waypoints.

[5] RAGI stands for Reliable Adaptive Game Intelligence.

[6] RETALIATE stands for REinforced TActic Learning In Agent-Team Environments.

From a designer's perspective, adapting can be dangerous because it surrenders control to a process that may not be fully understood. An adapting agent may develop strange or poor behaviour that violates intended design. For example, an agent originally intended to engage opponents may learn to hide from them as it adapts to avoid danger. Still, an adapting agent has the potential to discover intelligent new techniques that surprise its opponents by extending its original programming.

2.1.4 Case Study: The Counter-Strike: Source Bot

Individually, most of the techniques described above do not constitute a complete, working system. A complete, autonomous combat agent is a large undertaking, often involving interplay between preprogramming, planning, and machine learning techniques to achieve acceptable performance. An example of such a system is the official computer-controlled opponent (or *bot*) designed for Valve's *Counter-Strike: Source* [2004], (or *CSS*). Booth [2004] contributes a discussion of the development of this critically acclaimed agent.

The *CSS* bot is preprogrammed with a *navigation mesh*[7] enhanced with various annotations.[8] These annotations include whether a particular segment of the mesh is a valid hiding spot, and whether it must be jumped across or crouched through. When isolated from teammates, the *CSS* bot moves slowly to generate less noise. The bot is also preprogrammed with an awareness of *approach points* (i.e., directions from which enemies can appear) and employs realistic view control to monitor these locations.

The *CSS* bot also incorporates planning. Its future actions are determined using *scenario objectives*, or goals, that direct its future actions conditioned upon the current game state. To plan efficient paths through the environment, the bot incorporates the time penalties associated with having to jump or crouch over particular segments of the navigation mesh.

The *CSS* bot's adaptive behaviours include its annotation of the navigation mesh with a decaying *danger* level that increases the perceived cost of entering an area. The bot's *morale* increases when it is successful in play; high morale increases a bot's preference for rush tactics. These measures ensure variable behaviour across subsequent rounds of play.

[7] A navigation mesh is similar to a waypoint graph, except that it specifies a discretisation of the entire environment into non-overlapping regions. Its possible uses are identical to those described in Section 2.1.1.

[8] In new environments, the agent autonomously explores and annotates the environment on its own.

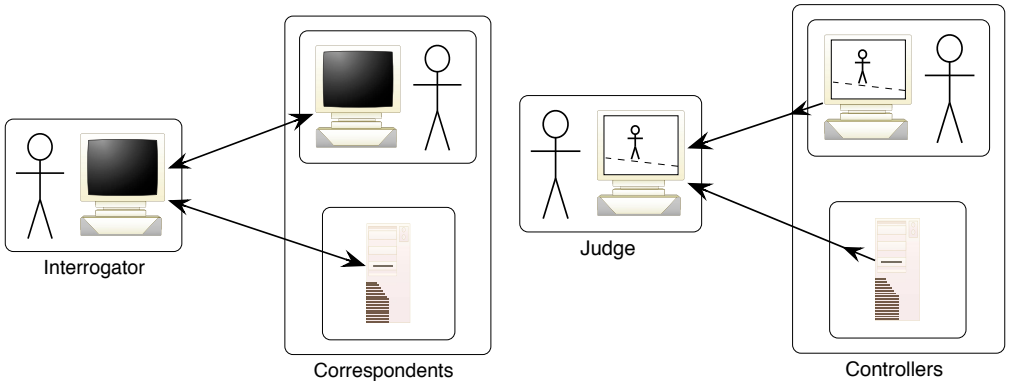


Figure 2.4: Experimental set-up for the Turing Test, shown left, and for believability testing, shown right. The Turing Test involves two-way communication between the interrogator and correspondent. In believability testing, the judges are restricted to making observations.

2.2 Believability Testing in Games

Human-likeness, or believability, is often emphasised over an accurate underlying model of human cognition, both in designing tactical characters for computer video games and training simulations [Wray *et al.*, 2004; Livingstone, 2006]. Indeed, many of the techniques described in the preceding section are intended to produce 'believable' behaviour. However, Gorman *et al.* [2006a] point out, "a significant impediment to work in this field is the lack of a formal, rigorous standard for determining how 'humanlike' an artificial agent is." **Believability testing** [Livingstone and McGlinchey, 2004] studies attempt to objectively measure the human-likeness of agents acting in virtual environments.

Believability testing resembles the Turing Test, proposed by Turing [1950] as a measure of machine intelligence. The original Turing Test tasks a human interrogator with judging whether a correspondent is human or machine via communication across a plain-text terminal; artificial agents that fool the interrogator are said to have passed. Believability testing similarly involves tasking human **judges** with rating the believability of an agent, but the judges can only observe the agent acting in an environment (i.e., communication is one-way, as illustrated in Figure 2.4). The following paragraphs describe existing research into objectively measuring agent believability in games.

Laird and Duchi [2000] investigate how changing individual behavioural parameters, *ceteris paribus*, affects the believability of the SOAR *Quakebot* (*q.v.* Section 2.1.2). The

study suggests trends in the importance of an agent's aiming skill (the precision with which the agent can target and anticipate its opponent's movements), decision time (an 'allowance' toward the number of computational cycles an agent may spend on planning per unit of time) and tactical complexity (the range of different tactical behaviours enacted by the agent during play). In particular, an agent given especially high aiming skill, especially low decision time, or scarce tactics may have lower believability. Unfortunately, the study only surveys eight judges and is considered preliminary by its authors.

Livingstone and McGlinchey [2004] investigate the respective believability of a human, a preprogrammed agent designed to appear human, and an imitation learning agent trained on human data in the classic video game *Pong*. The judges were on average unable to discern between humans and computer-controlled Pong bats. Some judges were, however, able to detect idiosyncracies among the players (i.e., "jerky and sudden movements") but not all of them understood its connection to the imitation learning agent. Livingstone and McGlinchey's observations highlight the subjective (and not necessarily accurate) perception of human judges; the authors state that, "where different observers can have quite different expectations, attempting to measure humanness may be fatally flawed."

Gorman *et al.* [2006b] investigate the believability of imitation learning agents trained on the movement of human players in Quake II. Players' overall choice of paths are modelled conditioned on their current power-ups (and, implicitly, which power-ups they likely desire). On average, the judges were unable to discern between humans and computer-controlled agents. As in Livingstone and McGlinchey's study, the judges detected idiosyncratic behaviour, citing "unnecessary jumping" and "[shooting] for no reason" as hallmarks of human behaviour. However, while the player movement in Quake II is significantly more complex than in Pong, the subjectivity of the human judges must be weighed appropriately.

2.3 Computing Openings

After Lincke [2000], I divide the different kinds of opening analysis that has taken place using computers into two categories. I distinguish between **active book construction**, wherein a computer explores a game's state-space to discover and evaluate openings, and **passive book construction**, wherein a computer analyses existing examples of gameplay.

Active book construction involves tasking a computer with building and evaluating openings through its own internal representation of a game. A simple example of this might be having an agent simulate play through all of the lines in a game of tic-tac-toe. From this experience, the program could determine which action sequences lead to losses, wins, and draws. This approach is possible in tic-tac-toe—and many other classic games—because the state of the game has an unambiguous representation and simple, well-defined rules.

Research in active book construction may address how to best explore very large game trees to create better opening books [Buro, 1999; Lincke, 2000], or how to develop complete theoretical solutions over game trees. For instance, Selby [1999] contributes an optimal solution[9] to the opening (or *pre-flop*) of Texas Hold'em Poker, which Billings *et al.* [2003] utilise in creating a strong poker-playing agent, *PsOpti1*. Tesauro's [1995] *TD-Gammon*, a reinforcement learning backgammon player, developed strong positional judgment that led it to using unconventional opening plays that have since been adopted in human tournament play. Both Yang *et al.* [2001] and Hayward *et al.* [2005] have determined full solutions to the game of Hex on reduced-size boards. And, more recently, Schaeffer *et al.* [2007] contribute a full theoretical solution to checkers, the largest of such games to be solved to date. All of these techniques have the advantage of not learning directly from human expertise; they are free to develop techniques on their own, and without bias. However, in general, these techniques are strongly reliant upon an unambiguous, easily represented state space. As described in Section 1.3, this is atypical of computerised combat simulations.

Passive book construction involves the analysis of *existing* examples of play to discern effective openings from ineffective openings. Passive book construction may be a viable alternative to active book construction, especially in the absence of a clear or tractable way to perform a search of a game's state-space. For instance, chess is an example of a classic strategy game for which deep online search is restricted by a massive state space and the real-time demands of tournament play. Hyatt [1999] looks into maintaining and evaluating the openings in a large database of games to equip his chess-playing agent *Crafty* with a rich library of openings as played by humans. An inherent challenge in maintaining such a large database is that not all of the openings are necessarily good (humans often blunder) and

[9] Given a fully randomised rollout of cards after the pre-flop occurs, with no further betting.

moreover not all of the openings are compatible with Crafty's playing strengths. As a result, Crafty evaluates openings by playing them. Hyatt points out that an opening associated with a win is not as 'good' as an opening associated with a loss is 'bad' because of how frequently human players blunder, especially under time pressure. This may also be true in computerised combat simulations, whose real-time demands put players under duress. Campbell *et al.* [2002] describe how the chess-playing computer program *Deep Blue* makes use of a similar database of past games. Deep Blue assigns bonuses and penalties to the openings available to it in its 'extended book' database, not by experience, but according to fixed criteria; these include an opening's frequency of play, the ratings of players that used the opening, the opening's historical recency, its rate of success in play, and past human expert judgment on the quality of a move. In one of Deep Blue's 1997 games against Gary Kasparov, this extended book helped Deep Blue to "[establish] a very comfortable position from the opening" [Campbell, 1999].

2.4 Modelling Agent Motion

Modelling the movement of goal-directed agents is an active area of research with obvious application to computerised combat simulations. Specific to the context of real-time strategy games, Southey *et al.* [2007] employ Bayesian inference, hidden semi-Markov models, and abstraction to probabilistically track agents that have only been partially observed but have known motion models. An example of the type of problem that can be reasoned about under Southey *et al.*'s framework is as follows: upon witnessing a brief glimpse of an agent crawling through the bushes, one may wish to determine a probability distribution over the different starting and ending points that the agent is travelling between.

Ratliff *et al.* [2006] contribute maximum margin planning (or MMP), an imitation learning algorithm that automatically associates action costs with features based on examples generated by an intelligent model agent (or *expert*). For example, one feature may be the expert's distance from the wall when approaching a goal.[10] If the expert stays close to walls, an MMP learner will infer an increased cost for moving away from walls while attempting similar tasks. However, MMP relies on an *a priori* definition of suitable features, which can

[10]This example is due to my colleague Jeffery Grajkowski.

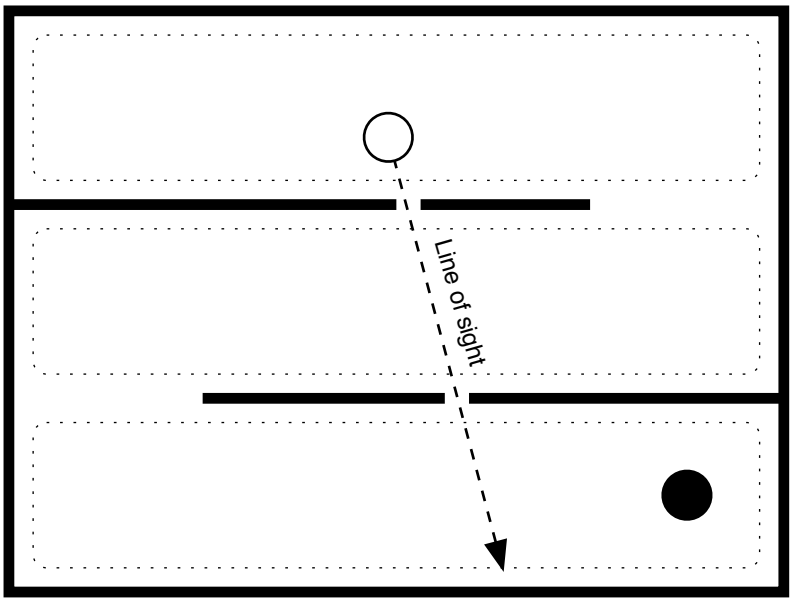


Figure 2.5: An agent (the filled circle) sneaks up on a sentry (the open circle). An expert in this environment would crawl whenever passing through the sentry's line of sight. This feature is produced by the environment's geometry and can be hard to represent; other tactical 'features', some incorporating temporal events, can be even harder to represent.

be difficult to provide. In particular, some of the advantage points in computerised combat simulations can be hard to 'featurise.' A simple example of this is illustrated in Figure 2.5, which shows an example of features that expert players recognise but can be hard to define.

Bererton [2004] describes a novel spectrum of modelling techniques for efficiently solving multi-agent Markov decision processes (MDPs). These techniques range from describing such decision problems using linear programs (whose solution can be efficiently computed) to increasingly expressive mixed integer program representations. Bererton's solution techniques are shown to be effective and efficient on a variety of tasks, and contribute to a framework for solving so-called *team-competition* problems wherein one team of agents chooses a cost function over an MDP which the competing team is tasked with solving. In particular, Bererton shows his solution techniques to be applicable to robotic control in a simplified game of paintball where unarmed robots attempt to navigate past armed sentries.

Gorman *et al.* [2006a] contribute a technique for imitation learning in Quake II. Their approach is to develop a waypoint graph over the space in which combat occurs via K-means clustering, and to use an inverse reinforcement learning algorithm to reward agents

for following the same paths as a human player.[11] As the human player's path develops, the reward associated with the waypoints he visits increases, which has the effect of eliminating from the agent's repertoire backtracking, local zigzags, and loops—behaviours that are probably unimportant Quake II, but are conceivably important in general tactical settings.

2.5 Summary

This chapter surveyed related work in the following areas: computationally-driven combat behaviour in computer simulations, including video games and training simulations; agent believability testing, which aims to objectively measure agent realism; the computational analysis of game openings, including *active* and *passive* book construction; and the modelling of goal-directed agent motion. While revealing of a number of significant techniques and discoveries in each area, this survey also shows that no prior work has specifically addressed the problem of establishing well-defined opening theory for tactical shooters.

[11] Another machine learning algorithm based on Bayesian inference learns which actions—turning, shooting, etc.—to perform as the agent moves.

Chapter 3

Trajectories

Alice said, "Would you please tell me which way to go from here?"
"That depends a good deal on where you want to get to," said the Cat.

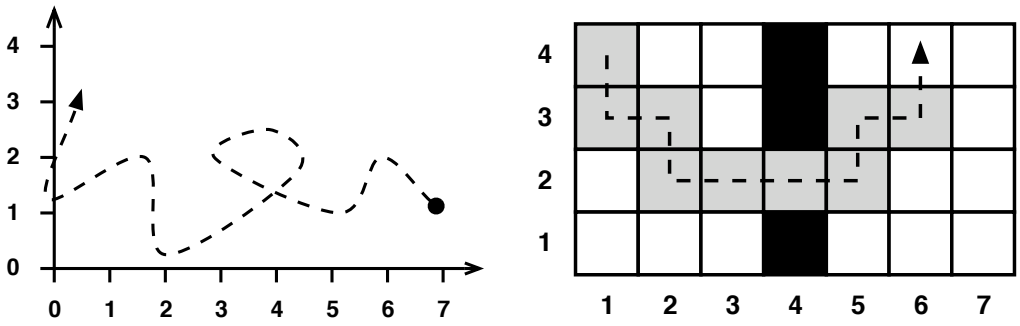
– Lewis Carroll's *Alice in Wonderland*

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 computerised combat simulation.

This chapter provides a general formalism for *spatial* trajectories being observed at discrete intervals. These concepts will serve as the core unit of analysis in the chapters that follow. Here, the reader will be introduced to a *notation* with which to represent trajectories, a definition of the *distance* between trajectories, and simple *operations* with which to manipulate trajectory data. This chapter concludes with a summarising example.

3.1 Defining Trajectories

Though trajectories are easy to grasp intuitively using diagrams such as those in Figure 3.1, a formal definition and a notation with which to represent them is required. The formal definition is precise, and a consistent notation will be helpful in describing algorithms that manipulate trajectories.



(a) A loop in $\mathbb{R}^2$ (b) A grid-world trajectory

Figure 3.1: Intuitive examples of trajectories.

Definition 3.1.1. A **discrete trajectory** X is a time-indexed set of points, $\{X_t : t \in T\}$, where:

- Each $X_t \in \mathbb{R}^n$ is a point in n-dimensional space, occurring at time-step t;
- $T \subseteq \{\mathbb{Z} > 0\}$ is a set of positive, integer-valued time indices.[1]

Note that, because each element in a trajectory is indexed by time, an object that remains stationary for n time-steps will still be described by a trajectory containing n points.

If there is a constant difference between the adjacent indices in T, I will call X a **constant-interval trajectory**[2]—otherwise I will call X a **variable-interval trajectory**.[3] Finally, since trajectories are defined by a set of time-indexed points, I will use the notation $\emptyset$ to denote the **empty trajectory** (i.e., a trajectory that contains no points).

Example 3.1.1. The following are a few examples of discrete trajectories:

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

[1] When a trajectory is implicitly defined (e.g., by a mathematical equation or another continuous, ongoing process), then $T = \{\mathbb{Z} > 0\}$.

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

[3] 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.

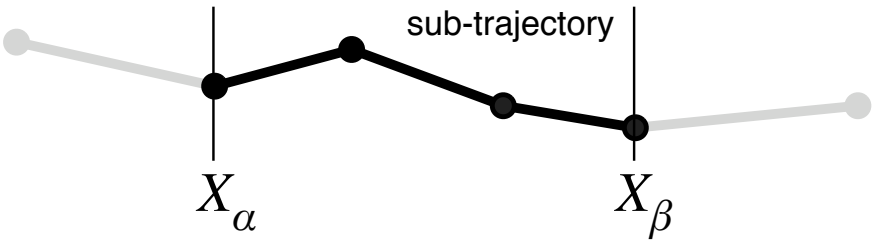


Figure 3.2: Illustration of a sub-trajectory.

In some instances it may be desirable to refer to a sequence of contiguous points occurring *within* a trajectory (Figure 3.2). This need motivates the following definition:

Definition 3.1.2. Let $X = \{X_t : t \in T\}$ be a trajectory. A **sub-trajectory** of X from α to β, where $0 < \alpha \leq \beta$, is defined as follows:

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

Example 3.1.2. 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 $Y = \{(1, 4, 0)_1, (3, 2, 0)_3, (5, 3, 0)_8, (6, 4, 0)_{10}\}$,
  - 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 $Y\Big|_{8}^{8} = Y_8 = \{(5, 3, 0)_8\}$.

The definitions introduced in this section provide a basis for reasoning about and manipulating individual trajectories. Since this study focusses on the analysis of a large volume of trajectory data, a definition of how two trajectories differ from each other is required.

3.2 Distance

Distance is important in any classification or pattern recognition task, as well as in cluster analysis wherein objects are grouped into related clusters. Indeed, distance plays a central role in my study: because there are many individual variations over what players perceive to be the 'same' path in computerised combat simulations, the question of whether two paths are equivalent is one of distance.

Several techniques exist to determine the distance between two trajectories. In general, the appropriateness of different distance measures for trajectory data is *highly* domain-dependent. At one extreme, one can measure the distance between the individual line segments that make up trajectories, as considered by Lee *et al.* [2007]. But the necessary pre-processing risks removing important points from a trajectory.[4] Alternatively, one can measure distance by interpolating between points, as considered by Yanagisawa *et al.* [2003]. But this approach runs the risk of introducing *superfluous*[5] points to a trajectory. This is not to say that neither of these techniques is applicable. However, taking the simple Euclidean distance between trajectories avoids these risks, is familiar, and is shown by Keogh and Kasetty [2002] to be superior to a number of specialised competing distance metrics on common time series (i.e., one-dimensional trajectory) datasets. This leads to the following definition of distance.

Definition 3.2.1. Let X and Y be discrete trajectories through n-dimensional space such that $X = \{X_t : t \in T\}$ and $Y = \{Y_t : t \in T\}$. (In general, let X_t^d refer to X_t's coordinate value along the d'th dimension.) The **distance** between X and Y is defined as follows:

$$\mathrm{dist}(X, Y) = \sqrt{\sum_{d=1}^{n} \sum_{t=1}^{|T|} \left(X_t^d - Y_t^d \right)^2}. \tag{3.1}$$

For notation convenience, the following denotes the distance between two sub-trajectories:

$$\mathrm{dist}(X, Y)\Big|_{\alpha}^{\beta} = \mathrm{dist}\left(X\Big|_{\alpha}^{\beta}, Y\Big|_{\alpha}^{\beta} \right) \tag{3.2}$$

[4] Pre-processing to remove points along which a trajectory does not change rapidly, according to the information theoretic *minimum description length* principle employed by Lee *et al.* [2007] may help to distill important line-segment information, but subtle changes along a trajectory can be tactically vital.

[5] Trajectories recorded at discrete time intervals risk omitting local zigzags [Chandrasekaran *et al.*, 2002] that correspond to important terrain (strategically, physically, or otherwise).

3.3 Operations on Trajectories

My eventual goal is to present algorithms that manipulate the discrete trajectories that arise from character movement in computerised combat simulations. This section defines and introduces notation for a variety of operations on trajectories that will enable these forthcoming algorithms to be described with precision. This section covers the addition, subtraction, scalar multiplication, and concatenation of trajectories. With the exception of concatenation, these operations are only defined for constant-interval discrete trajectories containing the same number of points taken at the same time indices.

The addition, subtraction, and scalar multiplication operations on trajectories are analogous to operations of the same name over vector data. Given a group of trajectories, these operations are necessary in tasks such as determining the group's arithmetic mean, or deriving a linearly weighted combination of the trajectories in the group, ideas that are explored more in the next chapter.

Definition 3.3.1. Let X and Y be discrete trajectories, both containing m points along the same time indices. That is, let $X = \{X_t : t \in T\}$ and $Y = \{Y_t : t \in T\}$, where $|T| = m$. The **addition**, **subtraction**, and **scalar multiplication** of trajectories are defined as follows:

- **Addition**: $X + Y = \{Z_t : Z_t = X_t + Y_t, t \in T\}$.
- **Subtraction**: $X - Y = \{Z_t : Z_t = X_t - Y_t, t \in T\}$.
- **Scalar Multiplication**: $cX = \{Z_t : Z_t = cX_t, t \in T\}$.

Example 3.3.1. Let X and Y be discrete trajectories, $X = \{(0,0)_1, (10,10)_2, (20,20)_3\}$ and $Y = \{(10,10)_1, (15,10)_2, (20,10)_3\}$. The following examples illustrate the use of the operations described by Definition 3.3.1:

1. $X + Y = \{(10,10)_1, (25,20)_2, (40,30)_3\}$.
2. $X - 3Y = \{(-30,-30)_1, (-35,-20)_2, (-40,-10)_3\}$.

The concatenation operation can be visualised as connecting two pieces of rope at the ends (Figure 3.3). This operation is useful for describing algorithms that construct trajectories in an incremental fashion (i.e., point by point). An example is shown in Section 3.4.

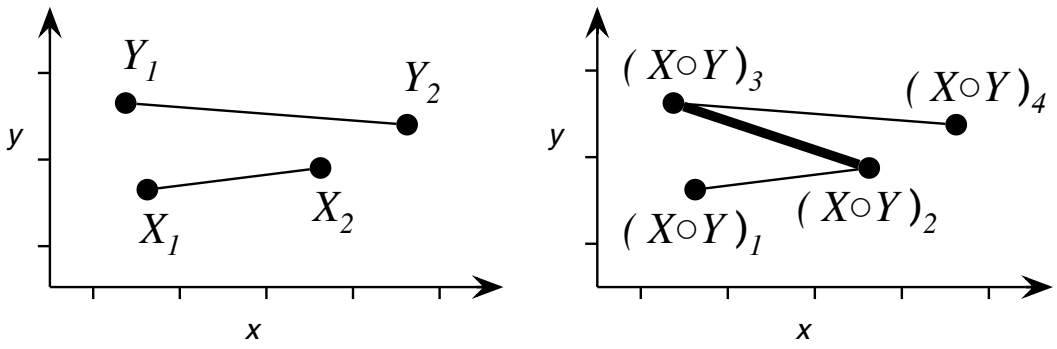


Figure 3.3: Illustration of the result of concatenating two trajectories in $\mathbb{R}^2$.

Definition 3.3.2. Let X and Y be two arbitrary discrete trajectories, $X = \{X_t : t \in T\}$ and $Y = \{Y_s : s \in S\}$. The **concatenation** of X and Y, denoted $X \circ Y$, is the result of time-shifting the points in Y forward so that their time indices succeed the time indices of the points in X, then unioning the result with X:

$$X \circ Y = X \cup \{Z_i : Z_{i+\max(X)} = Y_i, i \in S\}.$$

Addendum: The empty trajectory $\emptyset$ is the **concatenative identity** for discrete trajectories (i.e., the concatenation of $\emptyset$ with any other trajectory X simply results in X):

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

Example 3.3.2. Let $X = \{(0,0)_1, (10,10)_2\}$ and $Y = \{(10,10)_1\}$. The following examples illustrate the use of the concatenation operation defined by Definition 3.3.2.

1. $X \circ Y = \{(0,0)_1, (10,10)_2, (10,10)_3\}$.
2. $Y \circ (Y \circ Y) = \{(10,10)_1, (10,10)_2, (10,10)_3\}$.
3. $X - (Y \circ Y) = \{(-10,-10)_1, (0,0)_2\}$.

3.4 Summarising Example

Although simple, the definitions, measurements, and operations introduced in this chapter provide the foundation for a variety of general algorithms that can be used to manipulate, scale, categorise, and model trajectory data, as the following chapters will demonstrate.

1. Initialise A to be the empty trajectory $\emptyset$, and the current time index t to 0.
2. For as long as the recording process is taking place,
   (a) Sample and scale by half values from the functions f and g:
   $$F \leftarrow 0.5 \cdot \{F_1 = f(t)\}.$$
   $$G \leftarrow 0.5 \cdot \{G_1 = g(t)\}.$$
   (b) Concatenate A with the result of adding the two scaled values:
   $$A \leftarrow A \circ (F + G).$$
   (c) Increment the time index:
   $$t \leftarrow t + \Delta_t.$$

Figure 3.4: Pseoducode for the *running concatenation* algorithm.

To provide practical summary of many of these concepts, Algorithm 3.4 describes a practical example wherein a running concatenation of the average of two functions, f and g, is kept. The algorithm has a simple design: at discrete time intervals of length Δ_t, samples are taken from each of f and g and scaled by half. Their sum is then appended onto the end of a discrete trajectory, A, as illustrated in Figure 3.5.

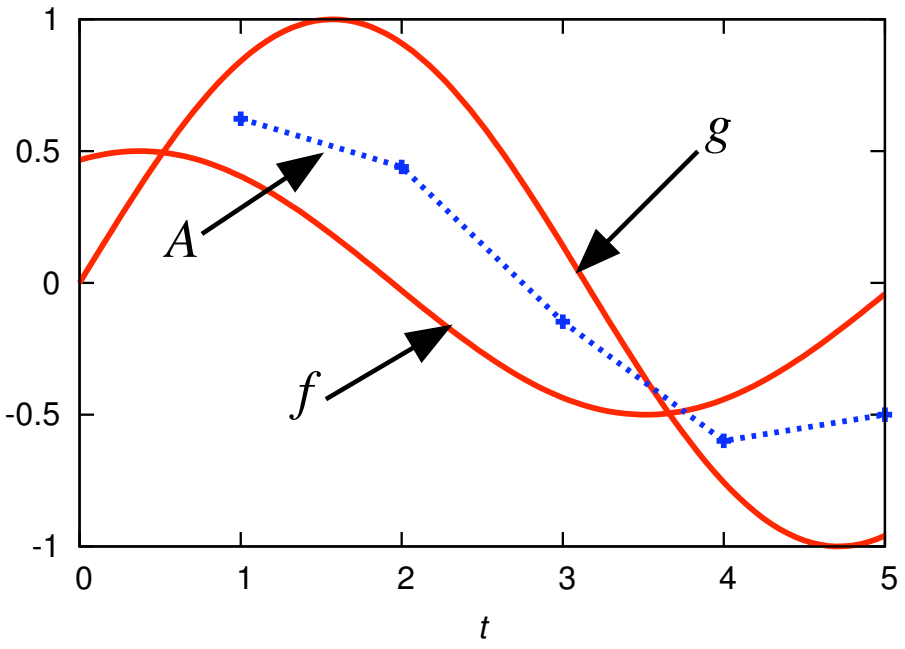


Figure 3.5: Illustration of the running concatenation algorithm.

Chapter 4

Problem Formulation

This chapter formalises the problem of discretising an opening space into categories that can be used to detect correlations between openings and game outcomes. I will further argue that an arbitrary discretisation of opening space is undesirable (i.e., each category should *model* a particular opening line), and describe empirical tasks to test whether a discretisation retains an accurate representation of the trajectories that players follow.

4.1 Discretisation

Recall that an individual player's *opening space* in a tactical shooter contains all of the openings that can be executed by that player over a fixed timespan starting from the beginning of the game. I argued in Section 1.3 that players do not treat an opening as a singular action sequence with a predetermined outcome, but rather as a category of many similar action sequences (i.e., there are *categories* of openings). I hypothesise that some categories of openings in tactical shooters are more likely to produce wins; this can be stated as follows:

> **Hypothesis**: *Discretising the opening space of a tactical shooter into regions of high similarity will reveal correlations between openings and game outcomes.*

Such a discretisation may be defined by a mapping. In particular, let Ω be an opening space, and $\mathbb{B}$ be a set (or "book") of categories of openings. Then, given the mapping:

$$\text{categorise} : \Omega \longrightarrow \mathbb{B}, \tag{4.1}$$

each $\mathbb{B}_i \in \mathbb{B}$ indexes an exclusive segment of opening space defined by the set:

$$\{\Omega_j \in \Omega : \text{categorise}(\Omega_j) = \mathbb{B}_i\}. \tag{4.2}$$

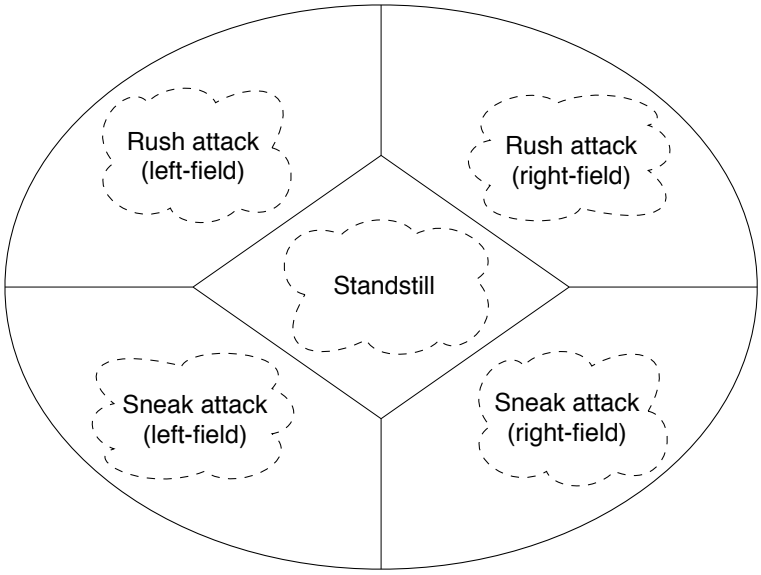


Figure 4.1: Illustration of a categorisation of an opening space. Categories are shown separated by solid lines. A mapping that defines a good categorisation has the effect of grouping similar behaviours and separating different behaviours, as shown.

With a $\mathrm{categorise}$ function, it becomes possible to tally game outcomes by category in a *contingency table* (Table 4.1). From here, testing the hypothesis that correlations exist between openings and game outcomes can formalised as a *chi-square test* for independence.

Ideally, given an effective $\mathrm{categorise}$ function, each $\mathbb{B}_i \in \mathbb{B}$ will index a set of conceptually similar openings (i.e., similar according to human intuition), as illustrated in Figure 4.1. A poorly designed $\mathrm{categorise}$ function can result in each $\mathbb{B}_i \in \mathbb{B}$ indexing a set of conceptually dissimilar opening behaviours, as illustrated in Figure 4.2. In the latter case, even if significant correlations were to be uncovered, the categories would not provide an intuitive explanation of the openings that were effective (or not) in play.

$\mathbb{B}_i$	Wins	Losses	Observations
$\mathbb{B}_1$	10	20	30
$\mathbb{B}_2$	10	10	20
Totals	20	30	**50**

Table 4.1: An example of a contingency table showing tallies of wins and losses under two categories of openings ($\mathbb{B}_1$ and $\mathbb{B}_2$). The rightmost column cells contain the total number of observations made for each of the two categories.

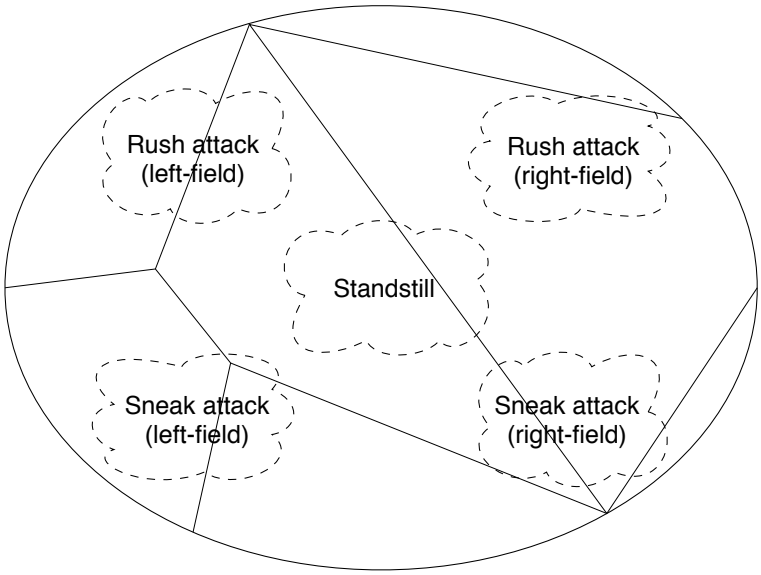


Figure 4.2: Illustration of a poor categorisation of an opening space. A poor categorisation can result in grouping conceptually different behaviours (such as *standstill* and *rush attack (left)*), and dividing conceptually similar behaviours (here, for example, the area corresponding to *rush attack (left)* is broken into two different categories).

The question of categorisation accuracy is important and warrants the formulation of an objective measure. The approach I explore is to treat a categorisation as a descriptive *model* of play (i.e., a representation that approximately explains how openings are carried out). This treatment enables the comparison of a categorisation to other (baseline) models of opening trajectories. The following sections describe empirical tasks towards this end.

4.2 Prediction Task

The **prediction task** provides an objective measure of how accurately a model may be used to forecast a partially completed opening trajectory. With reference to Figure 4.1, suppose a particular discretisation of opening space defines a category of left-field rush attacks. If a player is observed enacting half of a left-field rush, such a model might be used to infer this player's future trajectory (in this case, the fully completed left-field rush).

Any model that excels at the prediction task has potential practical application. In particular, prediction of a player's actions can be useful in creating adaptive computer-controlled opponents, or in creating teammates who learn complementary behaviours, or in generally developing more *believable* characters who play openings as humans do.

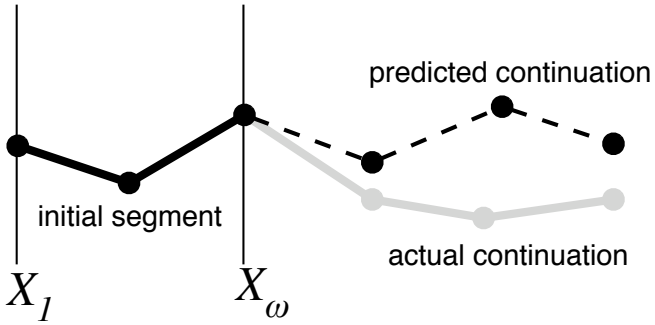


Figure 4.3: A sketch of the trajectory prediction task. Left, a trajectory's initial segment is shown, its points occurring between time indices 1 and ω. The trajectory's *actual* continuation after ω is shown in grey, and a *predicted* continuation, is shown with dashed lines. The predicted continuation is scored in terms of its distance from the actual continuation.

Formal Task Specification

Let X be a constant-interval discrete trajectory containing m points. The **initial segment** of X is the sub-trajectory from the initial point in X to an intermediate point $0 < \omega < m$:

$$X\Big|_1^\omega = \{X_t \in X : t \leq \omega\}.$$

The **actual continuation** of X from this initial segment is the remaining sub-trajectory:

$$X\Big|_{\omega+1}^{\mathrm{m}} = \{X_t \in X : t > \omega\}.$$

The objective of the prediction task is to recover a **predicted continuation**, C, that approximates the actual continuation of X. C and the actual continuation of X are trajectories of the same length (*i.e.*, $X\Big|_1^\omega \circ C$ has the same number of measurements as X). Specifically, given a training set $\mathbb{T}_{\mathrm{train}}$, and a test set $\mathbb{T}_{\mathrm{test}}$, a model is used to predict as follows.

1. **Training Phase**: A model $\mathbb{B}$ is constructed using the trajectories in $\mathbb{T}_{\mathrm{train}}$.

2. **Testing Phase**: Given the initial segment of each $\mathbb{T}_i \in \mathbb{T}_{\mathrm{test}}$, $\mathbb{B}$ is used to forecast C, a predicted continuation of $\mathbb{T}_i$.

A model $\mathbb{B}$'s performance at this task is measured in terms of the average error (i.e., distance) between $\mathbb{B}$'s predicted continuation, C, and $\mathbb{T}_i$'s actual continuation:

$$\mathrm{err} = \mathrm{dist}\left(C, \mathbb{T}_i\Big|_{\omega+1}^{|\mathbb{T}_i|}\right).$$

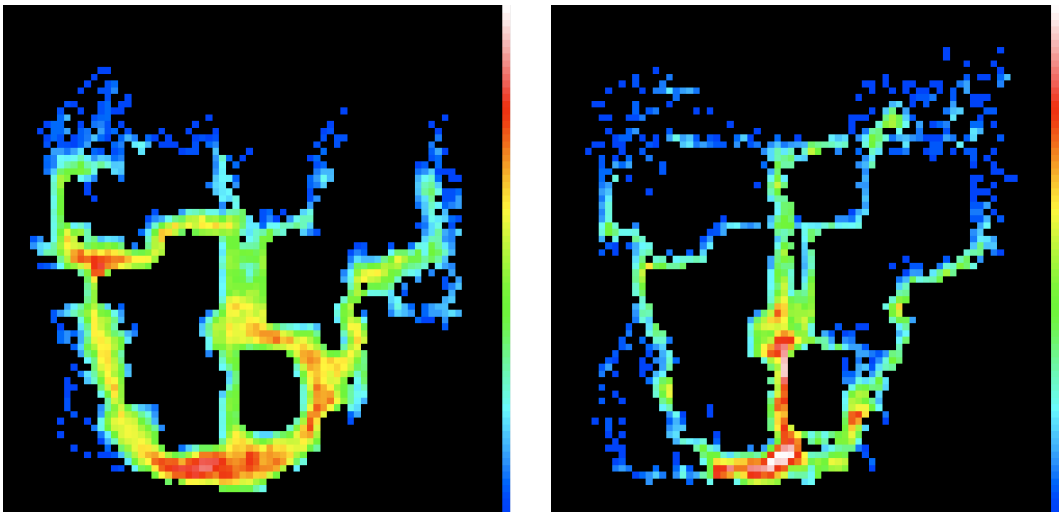


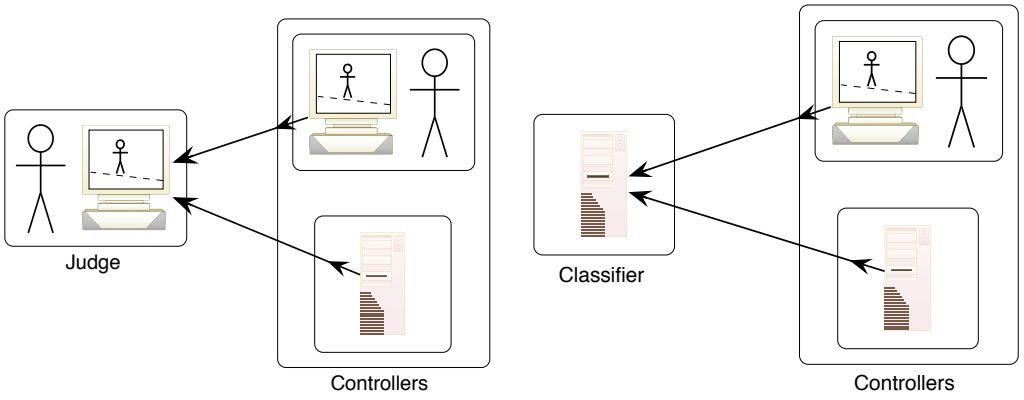
Figure 4.4: Heatmaps showing entropy in the moments prior to tactical conflict between teams in Valve's *Counter-Strike: Source*. This figure illustrates some of the differences between the decision making of humans (shown left) and bots (shown right).

4.3 Classification Task

The classification task measures how well the construction of a model retains the defining traits of two characteristically different groups of players. For example, suppose two groups of players both use a similar set of openings, but with idiosyncratic differences. In this study, these two groups are human-controlled characters and computer-controlled characters playing in Valve' *Counter-Strike: Source* [2004]. The characteristic differences can be visualised, as in Figure 4.4, in terms of the *entropy* with which players behave when acting during the game's opening.[1]

While primarily motivated as an objective measure of the accuracy of a categorisation of openings, models that excel in this task have practical application in other areas. For instance, fraud detection (i.e., determining whether or not a player is cheating or using an unsanctioned interface) may be possible through the recognition of certain kinds of play. Also, by providing a comparison of computer control and human control, this task covers new ground in believability testing: the convention is for *human* judges to rate the believability of an agent[2] (Figure 4.5(a)), but I know of no prior research in *programmatically* measuring believability (i.e., by automated means).

[1] See Appendix A.4 for details on how entropy values are determined.
[2] Related work in believability testing is discussed in Section 2.2.



(a) Believability testing.

(b) Classification task.

Figure 4.5: Experimental set-up for conventional believability testing, left, and the classification task, right. In conventional believability testing, a human judge observes a controller and rates its believability, or human-likeness. In the classification task, a mathematical model of play, stored on a computer, is used to classify controller type.

Formal Task Specification

Given a set of trajectories $\mathbb{T}_{\text{human}}$ created by human players, a set of trajectories $\mathbb{T}_{\text{bot}}$ created by computer-controlled characters, and a set of unlabelled test trajectories $\mathbb{T}_{\text{test}}$, a model's ability at the classification task can be empirically measured as follows:

1. **Training Phase**: A model $\mathbb{B}_{\text{bot}}$ is constructed based upon the trajectories in $\mathbb{T}_{\text{bot}}$, and a model $\mathbb{B}_{\text{human}}$ is constructed based upon the trajectories in $\mathbb{T}_{\text{human}}$.

2. **Testing Phase**: For each trajectory $\mathbb{T}_i \in \mathbb{T}_{\text{test}}$ the model that best describes $\mathbb{T}_i$ is used to label that trajectory.[3]

The mean frequency with which trajectories are correctly labelled is indicative of the classification power of the model, but also of the behavioural differences between human and computer players. A model that has been shown to excel at classification compared to other models can provide valuable, automated feedback to a designer concerning how significant the difference between human-controlled and computer-controlled play is.

[3]The mechanism for labelling differs with each model, as described in Chapter 5.

4.4 Summary

This chapter formalised the problem of distinguishing opening trajectories in combat simulations via a $\mathrm{categorise}$ function. This function maps trajectories to categories, and implies a *discretisation* of an opening space. This makes it possible to measure correlations between trajectories and game outcomes, which is essential for meaningful post-game analysis.

To provide a measure of how well a $\mathrm{categorise}$ function models the underlying distribution of trajectories, criteria (that it be able to *predict* and *classify* between different styles of play) were described and formalised as empirical tasks. These criteria will facilitate a comparison of a discretisation to other approaches to modelling trajectory data.

Chapter 5

Proposed Trajectory Models

Chapter 4 formulated the problem of discretising an opening space. This chapter introduces the approach of deriving *prototype* trajectories, via cluster analysis, from a set of example trajectories, $\mathbb{T}$. To put into context the accuracy with which these prototypes model $\mathbb{T}$'s generating mechanisms, two additional models are introduced: (1) a probabilistic (Markovian) model of transitions through discrete states, and (2) a model based on nearest neighbour estimates. Sketches of these three techniques are shown in Figure 5.1.

In the following sections, each technique is described in terms of its formal representation, its derivation from sample trajectory data, and definitions of corresponding the $\mathrm{predict}$ and $\mathrm{classify}$ functions that facilitate an empirical comparison of the three models.

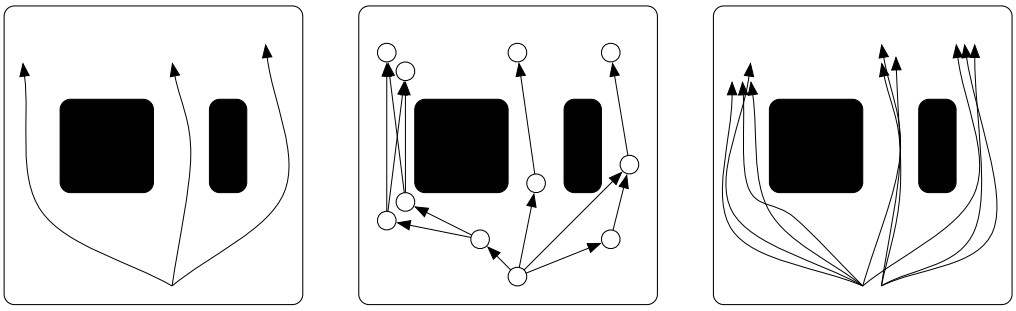


Figure 5.1: Sketches of different ways to model the generating mechanism for the trajectories being followed in a spatial environment. Left, the trajectories may be assumed to be following prototypes; middle, the trajectories may be thought of as a series of probabilistic transitions between states (illustrated here with circles); or, as shown right, all observed trajectory data can be retained to make generalisations using nearest neighbour estimation.

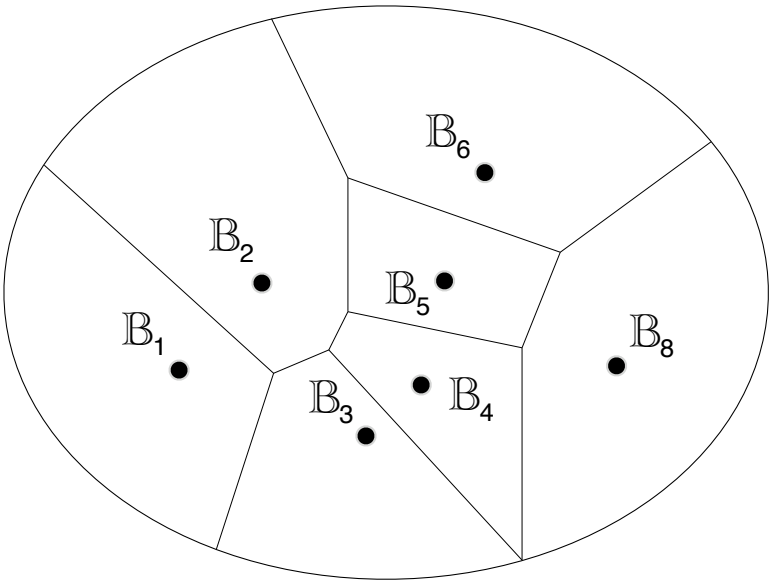


Figure 5.2: Illustration of a Voronoi tessellation.

5.1 Opening Book Approach

This approach is based on modelling trajectories with a set of **prototypes**.[1] The following subsections describe such a model's mathematical representation, its derivation via cluster analysis from a set of example trajectories $\mathbb{T}$, and definitions of the $\mathrm{predict}$ and $\mathrm{classify}$ functions that will facilitate an empirical comparison to the forthcoming baseline models.

5.1.1 Representation

Prototype modelling involves the definition of a set (or **opening book**) of K representative trajectories in an opening space, Ω. In general, any non-prototype trajectory (i.e., $\Omega_j \in \Omega \setminus \mathbb{B}$) is assumed to be an individual variation of the prototype to which it is nearest.

Definition 5.1.1. An **opening book** of trajectories is simply defined by the set $\mathbb{B} \subset \Omega$:

$$\mathbb{B} = \{\mathbb{B}_1, \mathbb{B}_2, \ldots, \mathbb{B}_K\},$$

As a means of discretising opening space, a trajectory $\Omega_j \in \Omega$ is categorised as:

$$\mathrm{categorise}(\Omega_j) = \underset{\mathbb{B}_i \in \mathbb{B}}{\mathrm{argmin}}(\mathrm{dist}(\mathbb{B}_i, \Omega_j)), \tag{5.1}$$

which defines a Voronoi tessellation of a state-space, as illustrated in Figure 5.2.

[1] When the number of prototypes is as large as the number of elements in the dataset, this approach is equivalent to nearest neighbour estimation with neighbourhoods of size one (see Section 5.3).

As stated in Section 4.1, tactical simulations accommodate countless variations over conceptually similar trajectories. This approach is hypothesised to lead to greater data efficiency by retaining a reduced set of prototypes that represent the complete dataset.

5.1.2 Derivation

I explore two approaches to deriving prototype trajectories via cluster analysis. These are the K-means clustering algorithm, a simple and popular prototyping approach to clustering, and the K-medoids clustering algorithm, a similar approach with potential benefits over K-means. Each of these algorithms are described in greater detail in Appendix A.

Approach 1: Derivation by K-means

K-means partitions a set of elements into spatially related subsets by iteratively refining a set of prototypes called centroids. To apply K-means clustering to trajectory data, the notion of the **centroid trajectory** of a set of trajectories must be defined.

Definition 5.1.2. Let $\mathbb{T}$ be a set of n discrete trajectories, $\mathbb{T} = \{\mathbb{T}_1, \mathbb{T}_2, \ldots, \mathbb{T}_n\}$, The **centroid trajectory** of $\mathbb{T}$ results from uniformly combining each $\mathbb{T}_i \in \mathbb{T}$:

$$C = \sum_{\mathbb{T}_i \in \mathbb{T}} \left(\frac{\mathbb{T}_i}{|D|} \right). \tag{5.2}$$

This definition enables the general K-means algorithm to be applied over trajectory data.

Approach 2: Derivation by K-medoids

The K-medoids algorithm strongly resembles the K-means algorithm. Here, however, prototypes must exist in the data (i.e., $\mathbb{B} \subseteq \mathbb{T}$), and are selected to minimise the *average* distance from points in the set. K-medoids can be less sensitive to outliers than K-means [Han and Kamber, 2006] and, in the context of trajectory data, guarantees that the prototype obeys the geometric constraints of the environment in which the set $\mathbb{T}$ was generated (Figure 5.3).

5.1.3 Prediction

Recall that the prediction task provides an objective measure of how accurately a model may be used to forecast a partially completed trajectory, X. Performing this task with a set of prototypes involves searching for the prototype with the closest initial segment to X.

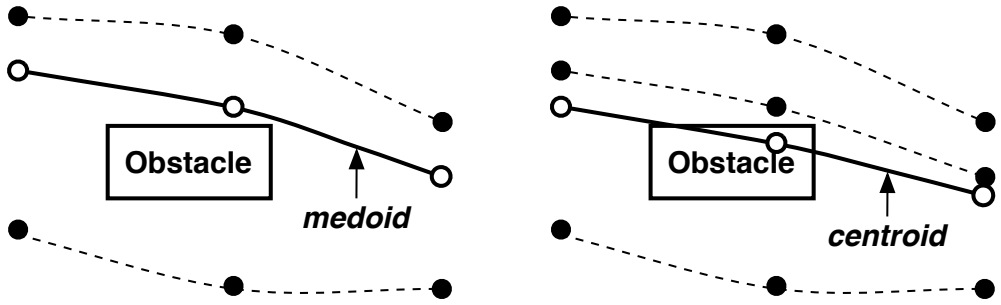


Figure 5.3: Comparative illustration of centroids and medoids in a constrained geometric environment. Left, the medoid of a set of trajectories will obey the geometric constraints of the environment in which the set was generated. This guarantee does not hold for a centroid over the same data, shown right, which may end up passing through obstacles or violating other strategic or environmental constraints.

Specifically, let X be a constant-interval discrete trajectory containing m points, and $\mathbb{B}$ be a set of K prototypes. The first step is to locate the trajectory $\mathbb{B}_i \in \mathbb{B}$ whose initial segment is closest to X. This is $\mathbb{B}_i = \underset{\mathbb{B}_i \in \mathbb{B}}{\operatorname{argmin}} \left(\operatorname{dist}(\mathbb{B}_i, X) \Big|_1^{\omega} \right)$. The next step is to determine the actual continuation of $\mathbb{B}_i$ up to time-step m, which is $\mathbb{B}_i \Big|_{\omega+1}^{m}$. Combining these steps provides the following definition of prediction using a prototype model:

$$\operatorname{predict}(X) = \underset{\mathbb{B}_i \in \mathbb{B}}{\operatorname{argmin}} \left(\operatorname{dist}(\mathbb{B}_i, X) \Big|_1^{\omega} \right) \Big|_{\omega+1}^{m}. \tag{5.3}$$

5.1.4 Classification

Recall that the classification task tests a model's ability to retain the characteristic traits of a group of players. Here, given a query trajectory X to classify, the idea is to determine which of two specialised prototype models contains the closest trajectory to X.

Specifically, let X be a constant-interval discrete trajectory, and $\mathbb{B}_{\text{human}}$ and $\mathbb{B}_{\text{bot}}$ be prototype models derived from separate datasets of human and bot trajectory data. For each $g \in \{\text{human}, \text{bot}\}$, the closest trajectory to X is found (this is $Y^g = \min_{Y \in \mathbb{B}_g} (\operatorname{dist}(X, Y))$. X is then associated with the model that produced the nearest Y^g:

$$\operatorname{classify}(X) = \underset{g \in \{\text{human}, \text{bot}\}}{\operatorname{argmin}} \left(\min_{Y \in \mathbb{B}_g} (\operatorname{dist}(Y, X)) \right). \tag{5.4}$$

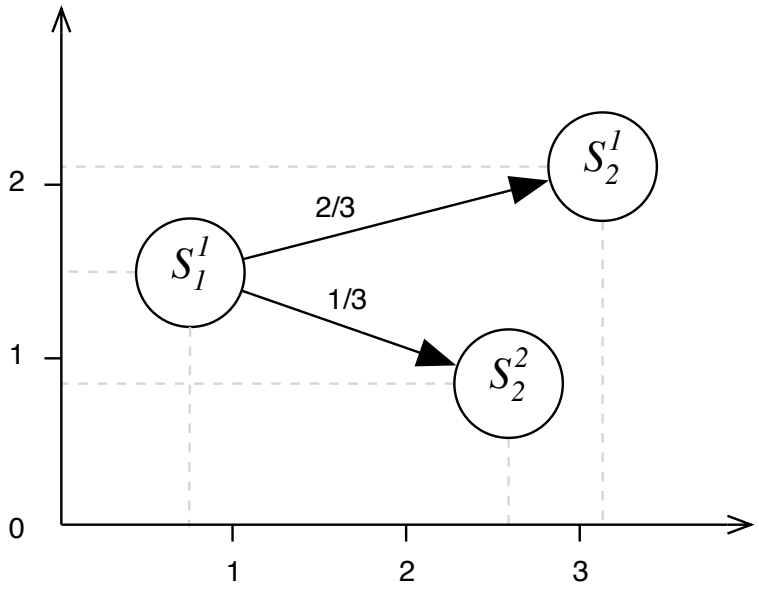


Figure 5.4: Illustration of a simple Markov model of trajectories with states represented by circles and transition probabilities represented by labelled arrows. Here, states are defined as $S = \{S_1 S_2\}$, where $S_1 = \{S_1^1\}$ and $S_2 = \{S_2^1, S_2^2\}$; the transition function P is defined as $P_1(1,1) = 2/3$ and $P_2(1,2) = 1/3$.

5.2 Markov Model of Trajectories

The Markov model provides a baseline for comparison against the prototype model's accuracy in modelling $\mathbb{T}$. It is based upon a Markov chain [Markov, 1971], which is a mathematical model of state transitions characterised by conditional independence of history, transitions over discrete time-steps, and stochastic transitions between states; these characteristics may lead to an appropriate representation of goal-directed trajectory data.

Specifically, the Markov model of trajectories is defined by time-organised states and the probabilistic transitions between them. The derivation of such a model from data involves clustering visitation locations at each time-step, the cluster centres defining states, and inferring transition probabilities between states based on the paths of trajectories.

5.2.1 Representation

A **Markov model of trajectories** (Definition 5.2.1) is a Markov chain with additional spatio-temporal information, as illustrated in Figure 5.4. Each state is situated in a par-

ticular point in space, and timing information is *explicitly* conveyed by subdividing the set of states based on time.[2]

Definition 5.2.1. A Markov model of trajectories is the tuple (S, P), where:

- $S = \{S_1, S_2, \ldots, S_q\}$ is a set containing *sets* of states subdivided by time index:
  - $S_i \in S$ is a set of states whose time index is i, and
  - $S_i^j \in S_i$ is a state with a location in $\mathbb{R}^n$,
- P is a transition function, where $P_t(i, j)$ gives the probability of transitioning from state S_t^i to state S_{t+1}^j. As an additional notational convenience, $P_t^{(n)}(i, j)$ gives the probability of transitioning from state S_t^i to state S_{t+n}^j over the course of n time-steps.

5.2.2 Derivation

Let $\mathbb{T}$ be a set of constant-interval discrete trajectories (Figure 5.5(a)). The derivation of a Markov model of trajectories occurs in two phases. In the first phase a set of states S is derived from $\mathbb{T}$; in the second phase a transition function P is inferred to define transition probabilities between each of the states in S.

Phase 1: Deriving States

The first phase in constructing a Markov model of trajectories is the derivation of states, S, from trajectory data. In this derivation, the number of states per time-step is defined *a priori*—the function $\kappa : \mathbb{N} \longrightarrow \mathbb{N}$ defines the number of states at time t.[3]

Let $\mathbb{T}$ be a set of trajectories, where an individual point occurring at time t along some trajectory $\mathbb{T}_i \in \mathbb{T}$ is denoted $\mathbb{T}_i(t)$. The process of deriving states from trajectory data is described in Algorithm 5.2.1: In step 1, the points occurring across all trajectories are divided by time of occurrence (Figure 5.5(b)). In step 2, K-means clustering is performed within the resulting sets (Figure 5.5(c)). Finally, in step 3, each S_t^i is defined by the centres of the resulting clusters (Figure 5.5(d)).

[2] Timing information can be implied by the transition function; it is explicit here for notational convenience.

[3] The function κ could be automatically derived using the *Gap statistic* [Tibshirani *et al.*, 2001]; in this study, however, κ is hand-selected.

Algorithm 5.2.1 Markov state derivation.

1. For t from 1 to m, create a set R_t containing all points that occur at time t:

$$R_t \leftarrow \bigcup_{\mathbb{T}_i \in \mathbb{T}} \{\mathbb{T}_i(t)\}.$$

2. For t from 1 to m, apply K-means clustering with $K = \kappa(t)$ to R_t to derive a set of clusters:

$$G_t = \{G_t^1, \ldots G_t^K\}.$$

3. For each G_t^i, define S_t^i to be a Markov state whose centre is the centroid of G_t^i.

Algorithm 5.2.2 Markov transition probability inference.

1. For t from 1 to m, let $G_t = \{G_t^1, \ldots, G_t^K\}$ be as derived in Algorithm 5.2.1.
2. Infer $P_t(i, j)$, the probability of a trajectory transitioning from each G_t^i to each G_{t+1}^j:
   (a) For each G_t^i, let T be the set of trajectories that pass through G_t^i:

$$T \leftarrow \{X \in \mathbb{T} : X_t \in G_t^i\}$$

   (b) For each G_{t+1}^j:
      i. Let V be the trajectories in T that also pass through G_{t+1}^j:

$$V \leftarrow \{X \in T : X_{t+1} \in G_{t+1}^j\}.$$

      ii. Define the transition probability with a frequentist ratio, $P_t(i, j) \leftarrow \frac{|V|}{|T|}$.

Phase 2: Inferring Transition Probabilities

The second phase of deriving a Markov model of trajectories lies in inferring the probabilistic links between the individual states derived in the Phase 1. The points along each trajectory in $\mathbb{T}$ can be mapped to one of the states defined in Algorithm 5.2.1 (Figure 5.5(d)), and in this way each trajectory adds to a record of state transitions over time. By observing the frequencies of these transitions, transition probabilities can be inferred by taking frequentist counts of the transitions between different clusters.

The process of inferring transition probabilities is described in Algorithm 5.2.2. Using the same set $\mathbb{T}$ and the clustering G that was used earlier to derive *states* (step 1), the algorithm counts the trajectories that pass through each cluster G_t^i at time t (step 2(a)). Next,

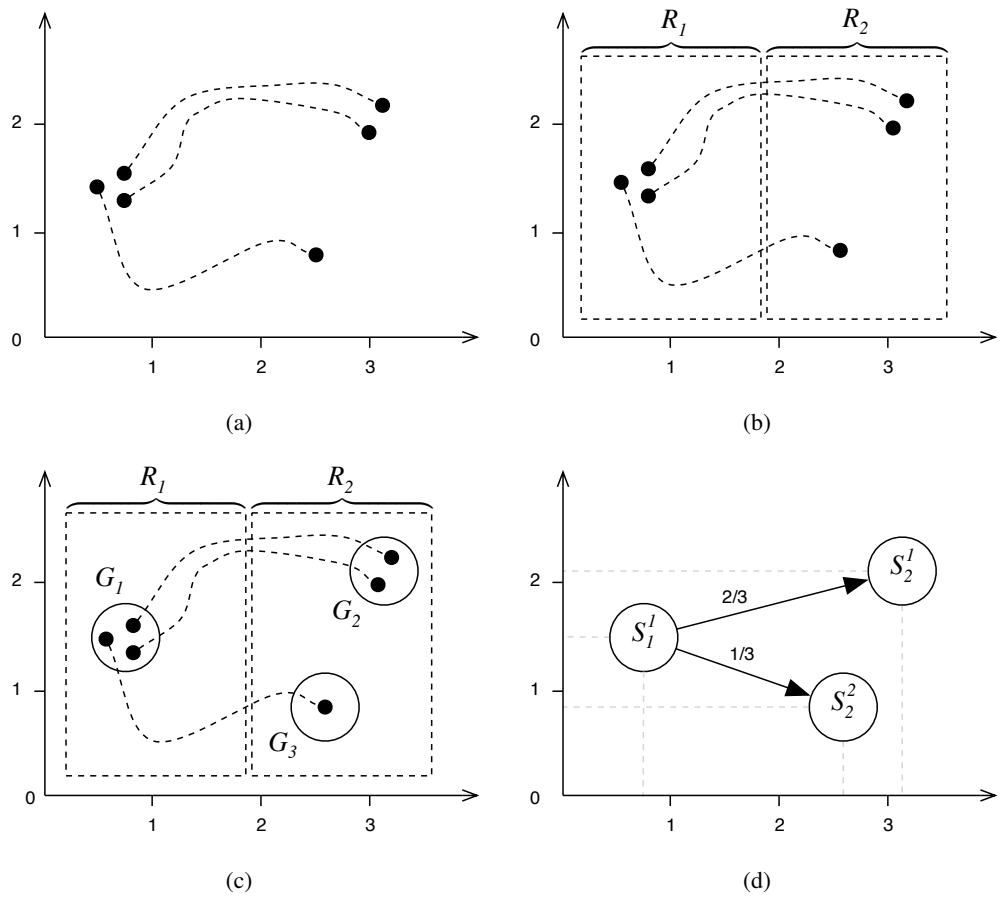


Figure 5.5: Constructing a Markov model, as described in Algorithms 5.2.1 and 5.2.2. Here, $\kappa(1) = 1$ and $\kappa(2) = 2$ (i.e., the derivation will create 1 state for the first time-step and 2 states for the second time-step).

the algorithm counts the number of trajectories that pass through G_t^i *and* G_{t+1}^j (step 2(b)). These two counts form a frequentist ratio that determines the probability of transitioning between states S_t^i and S_{t+1}^j.

5.2.3 Prediction

To predict future points from a trajectory's initial segment, a Markov model of trajectories can be used to define a probabilistically-weighted sequences of future points. Figure 5.6 illustrates how a trajectory that has been observed up to some nearby Markov state S can be forecasted as the probabilistically-weighted successors of S (i.e. $\sum_{S'} \big(P(S, S') \cdot S'\big)$). This process is formally described in Algorithm 5.2.3.

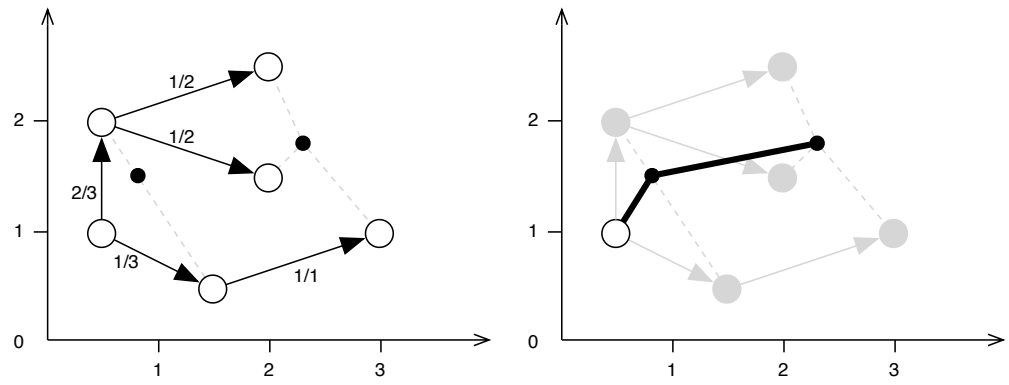


Figure 5.6: Predicting with a Markov model of trajectories involves building a predicted sequence of points by combining probabilistically weighted future points defined by the model (left), which are then concatenated into a predicted continuation (right).

Algorithm 5.2.3 Definition of $\text{predict}(X)$ for Markov models of trajectories

---

Require: A Markov model $\mathbb{M} = (S, P)$.
Require: κ is a function, $\kappa : \mathbb{N} \longrightarrow \mathbb{N}$

1. Locate the nearest state to X's terminal point at time ω:

$$i = \underset{i}{\text{argmin}}(S^i_\omega, X_\omega).$$

2. For t from 1 to $m - \omega$, iteratively construct a predicted continuation, R, by concatenating a probabilistically weighted sequence of the successors of S^i_ω:

$$R \leftarrow R \circ \left(\sum_{j=1}^{\kappa(\omega+t)} P^{(t)}_\omega(i, j) \cdot S^j_{\omega+t} \right).$$

5.2.4 Classification

Classification with Markov models of trajectories involves determining each model's likelihood of having generated a given query. Specifically, let X be a constant-interval discrete trajectory, and $\mathbb{M}_{\text{human}}$ and $\mathbb{M}_{\text{bot}}$ be Markov models of human and bot trajectories respectively. For each $g \in \{\text{human}, \text{bot}\}$ and each $X_t \in X$, let A^g_t indicate the nearest simultaneously-occurring Markov state defined by the model $\mathbb{M}_g$. X is then classified as:

$$\text{classify}(X) = \underset{g \in \{\text{human,bot}\}}{\text{argmax}} \left(\prod_{t=1}^{|X|-1} P^{(t)}_1(A^g_t, A^g_{t+1}) \right). \tag{5.5}$$

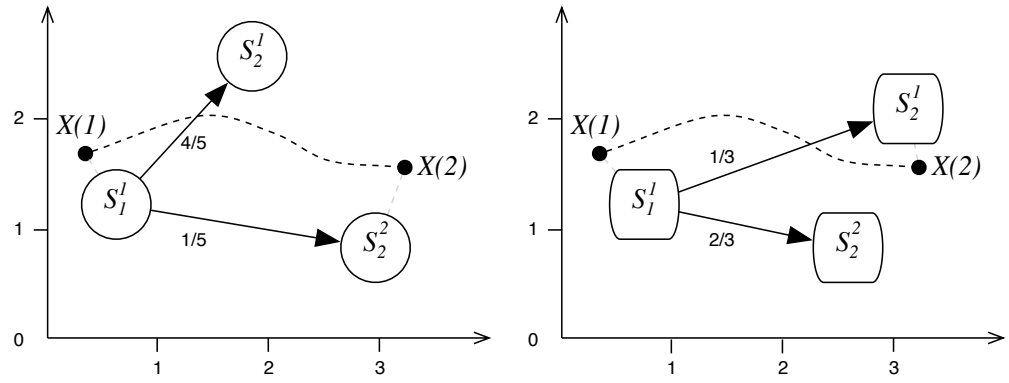


Figure 5.7: Classifying with a Markov model of trajectories is the process of determining which of two models is more likely to have produced a query trajectory X. On left, the superimposed model is determined to have a probability of $1/5$ of having generated X. The superimposed model on right has a probability of $1/3$ of having generated the same X.

5.3 Nearest Neighbour Estimation

The k-Nearest Neighbours (k-NN) model provides another baseline for comparison against the prototype model. Among the simplest of machine learning algorithms [Dasarathy, 1990], k-NN is characterised as a lazy learning technique because it postpones data generalisation until a query is made [Aha, 1990]. It has wide applicability, for instance, in recommender systems [Adomavicius and Tuzhilin, 2005], and typically performs well on a variety of practical tasks [Michie *et al.*, 1994].

In k-NN a query is generalised by looking at the neighbourhood of nearest points that surround it (i.e. a set containing k of the closest points according to a distance metric, as in Algorithm 5.3.1). This neighbourhood is then used to make claims about the query, sometimes using distance weighting to reduce the influence of very distant neighbours.

5.3.1 Representation

Definition 5.3.1. A **k-NN model** of trajectories is the tuple $(\mathbb{T}, k, , \varphi)$, where:

- $\mathbb{T}$ is a set of discrete trajectories,
- $k \in \mathbb{Z}^+$ is the size of neighbourhoods that will be expanded around a query trajectory,
- φ is a distance-weighting parameter.

Algorithm 5.3.1 k-NN algorithm.

Require: X is a query.

1. Initialise N to be the empty set: $N \leftarrow \{\}$.
2. Until $|N| = k$, add to N the nearest trajectory $\mathbb{T}_i \in \mathbb{T}$ not already in N:

$$N \leftarrow N \cup \left\{ \underset{\mathbb{T}_i \in \mathbb{T} \setminus N}{\operatorname{argmin}} (\operatorname{dist}(X, \mathbb{T}_i)) \right\}.$$

3. Return the set of neighbours N.

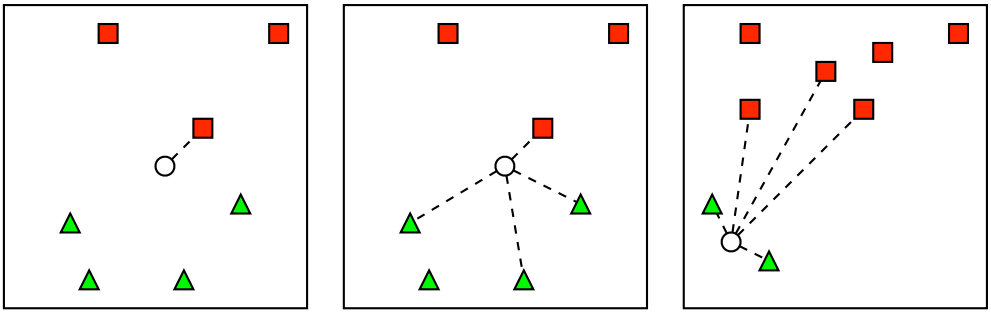


Figure 5.8: Examples of neighbourhoods using Euclidean distance in $\mathbb{R}^2$. The query is represented by an open circle, and the shape of the other points indicates their class. Shown left, a neighbourhood can consist of the nearest point to the query (1-NN), or of several nearest points (i.e., 4-NN, shown middle). Shown right, the makeup of a neighbourhood may not be as telling as the distance between the query and the points in its neighbourhood.

5.3.2 Prediction

Recall that the prediction task tests a model's ability to predict the continuation of a query trajectory X from its initial segment. Nearest neighbour estimates can be used to predict a trajectory's continuation by linearly combining the actual continuations of the trajectories whose initial segments neighbour the initial segment of X.

Specifically, let X be a constant-interval discrete trajectory of length m, whose initial segment is known up to time-step ω, and $\mathbb{K}$ be a k-NN model $\mathbb{K} = (\mathbb{T}, k, \varphi)$. X's neighbourhood, N, is found based on initial segments (Algorithm 5.3.2). Next, distance weights for each $T \in N$ are determined according to the distance-weighting parameter, φ:

$$\alpha(T) \leftarrow \frac{\operatorname{dist}(Q, T)^{-\varphi}}{\sum_{S \in N} \operatorname{dist}(S, Q)^{-\varphi}}. \tag{5.6}$$

Algorithm 5.3.2 Determining k-NN based on an initial segment of X.
Require: X is a query.

1. Initialise N to be the empty set: $N \leftarrow \{\}$.
2. Until $|N| = k$, add to N the nearest trajectory $\mathbb{T}_i \in \mathbb{T}$ not already in N:

$$N \leftarrow N \cup \left\{ \underset{\mathbb{T}_i \in \mathbb{T} \setminus N}{\operatorname{argmin}} (\operatorname{dist} \Big|_1^{\omega} (X, \mathbb{T}_i)) \right\}.$$

3. Return the set of neighbours N.

Finally, each $T \in N$ is linearly combined with weights specified by α:

$$\operatorname{predict}(Q) = \left(\sum_{T \in N} \alpha(T) \cdot T \right) \Bigg|_{\omega+1}^{m} \tag{5.7}$$

5.3.3 Classification

For classification a neighbourhood is derived from the combined trajectory data to perform a distance-weighted variation of a majority vote.

Let X be a constant-interval discrete trajectory, and $\mathbb{K}_{\text{human}} = (\mathbb{T}_{\text{human}}, k, \varphi)$ and $\mathbb{K}_{\text{bot}} = (\mathbb{T}_{\text{bot}}, k, \varphi)$ be Markov models of human and bot trajectory generators respectively. First, the data from both models is combined (i.e., $\mathbb{T}^h \cup \mathbb{T}^b$) and k-sized neighbourhood is derived from the combined data. Each trajectory votes with distance-weighting using its own label. The label with the highest tally is used to classify X:

$$\operatorname{classify}(Q) = \underset{g \in \{\text{human}, \text{bot}\}}{\operatorname{argmax}} \left(\sum_{T \in (N \cap \mathbb{T}^g)} \alpha(T) \right).$$

5.4 Summary

This chapter introduced an approach to discretising an opening space in a tactical shooter by means of prototype modelling. For validation of this 'opening book' by empirical analysis, two alternative models to serve as baselines were established, one based on states and probabilistic transitions, and the other based on nearest neighbour estimates. The next chapter describes an empirical study that will demonstrate that this proposed discretisation retains an accurate representation compared to the baselines.

Chapter 6

Empirical Study

This chapter describes an empirical study of the three models described in Chapter 5. The experimental platform is Valve's *Counter-Strike: Source* [2004], a tactical shooter in which strategy is vital. Following a brief introduction to Counter-Strike, this chapter describes results for the prediction and classification tasks, and results that demonstrate that a selective discretisation of opening space can reveal significant correlations between a human player's actions and the outcome of a game.

6.1 *Counter-Strike: Source* as an Experimental Platform

Counter-Strike: Source (*CSS*) is a computer video game that simulates combat in a spatial environment with obstacles, overpasses, ramps, areas of dark and light, and realistic physics. *CSS* ranks in the top ten of a 2007 Nielsen ranking of video games [Nielsen Media Research, 2007] and the franchise has been a subject of study in tactical behaviour [Manninen, 2001], network traffic analysis [Chang *et al.*, 2002], and ethnography [Vesterby, 2002].

6.1.1 Game Dynamics

CSS gameplay involves competition between two teams; generally, one team assumes the role of *attacker* and the other team assumes the role of *defender*. *CSS* focusses on realistic combat (tactical errors are heavily penalised) and calculated strategic play (superior strategy reliably leads to tactical dominance)—as shown in Figure 6.1, the loss of even a single teammate is strongly correlated with a reduced probability of winning a match.

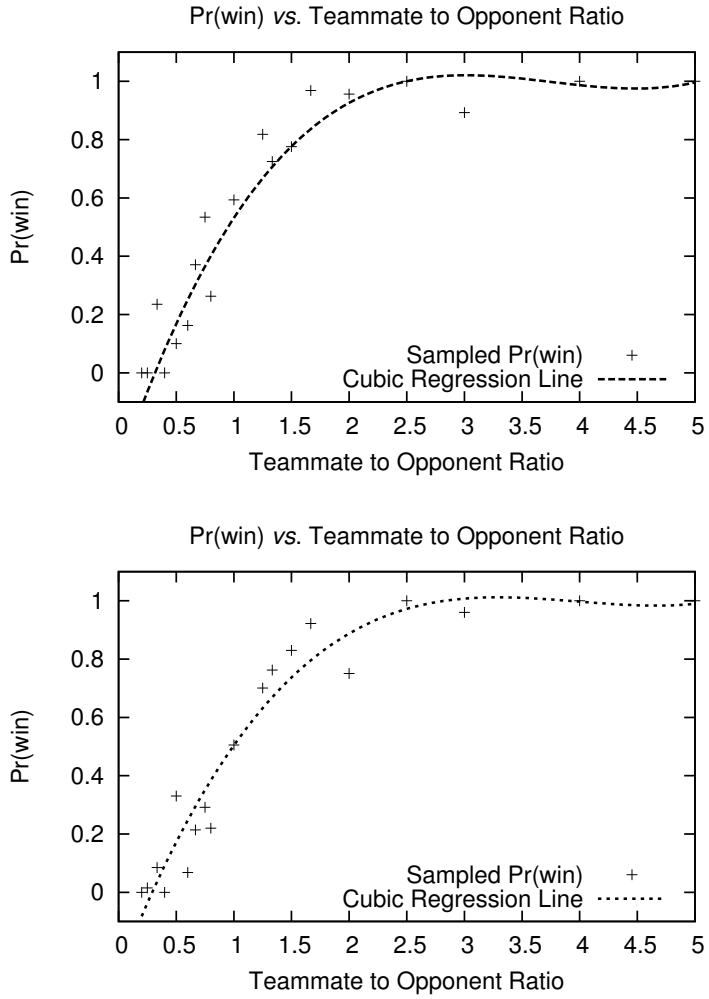


Figure 6.1: Plots showing the relationship between the ratio of teammates to opponents and the probability of winning in *dust2*. The graph on the top shows results for human gameplay; the graph on the bottom shows results for bot gameplay. The correlation is similar for both groups, and illustrates the significance of losing a teammate.

While not immersive in every respect,[1] *CSS*'s realism accommodates many real-world military tactics (e.g., ambushes, rushes, flanking maneuvers, and tactical formations). Environments are recycled across matches, which results in high player familiarity. Manninen [2001] describes the result as "highly developed playing routines, such as common tactics, specific roles of team members, and order of actions—in the extreme the game session gives out the feel of observing robots doing their tasks." The result is the repeated execution of specific openings, making *CSS* openings ideal for prototype modelling.

[1] Barlow and Morrison [2005] describe a variety of ways in which most combat simulations—including *Counter-Strike: Source*—diverge significantly from realism.

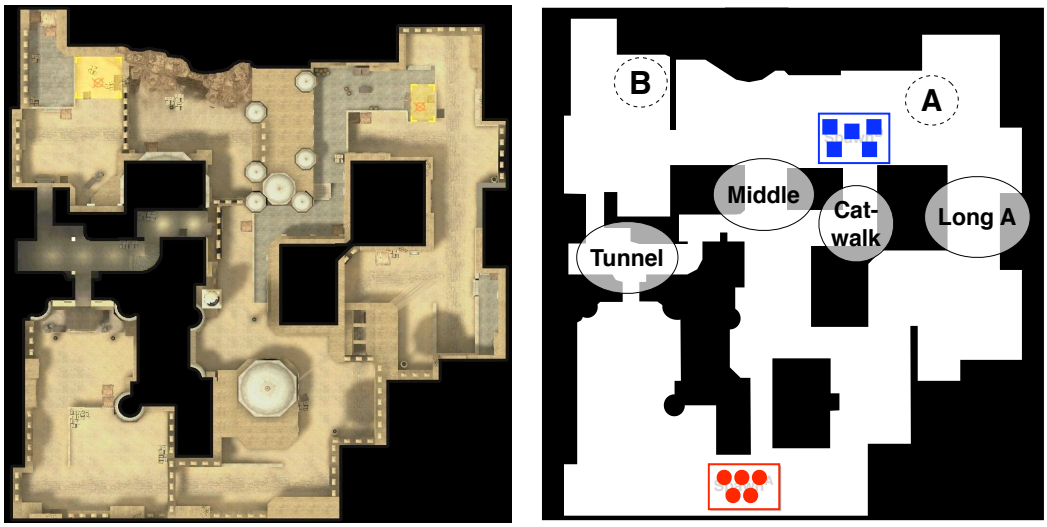


Figure 6.2: Overhead views of *CSS*'s popular *dust2* map. Left, the environment's graphical properties are illustrated. Right, labels indicating key landmarks and routes superimpose an illustration of the environment's geometric properties. The square region populated with five circles is the spawn area for attackers; the square region populated with five squares is the spawn area for *defenders*. The attacking team is tasked with attacking one of the sites labelled *A* or *B*, and the defending team is tasked with defending them.

Typical of the genre, game state information comes to each player through a situated view frustum that depicts the perspective of a specific game character (see Figure 1.3 on page 4). As a result, players rarely know the state of the world or the locations of other players, especially as the **opening phase** (Definition 6.1.1) of a game round develops.

Definition 6.1.1 (**Opening Phase**)**.** I define the opening phase of a *CSS* game round to be the time spanning from when players are first able to act to the moment tactical conflict occurs (i.e., a player is damaged or eliminated from play) or to a specified time limit.

This study focusses on the *CSS* virtual environment (or *map*) called *dust2*, a popular setting for tournament play (Figure 6.2). Attackers and defenders begin play within their respective **spawn areas** (Definition 6.1.2). The attacking team attempts to transport an explosive to one of sites A or B; the defending team tries to prevent this, and can win by defusing a planted explosive. Either team can win by eliminating all of the opposition.

Definition 6.1.2 (**Spawn Area**)**.** The spawn area is a region within which players are randomly placed when a game begins. The players of opposing teams are typically spawned far away from one-another, which allows for strategic development in the form of openings.

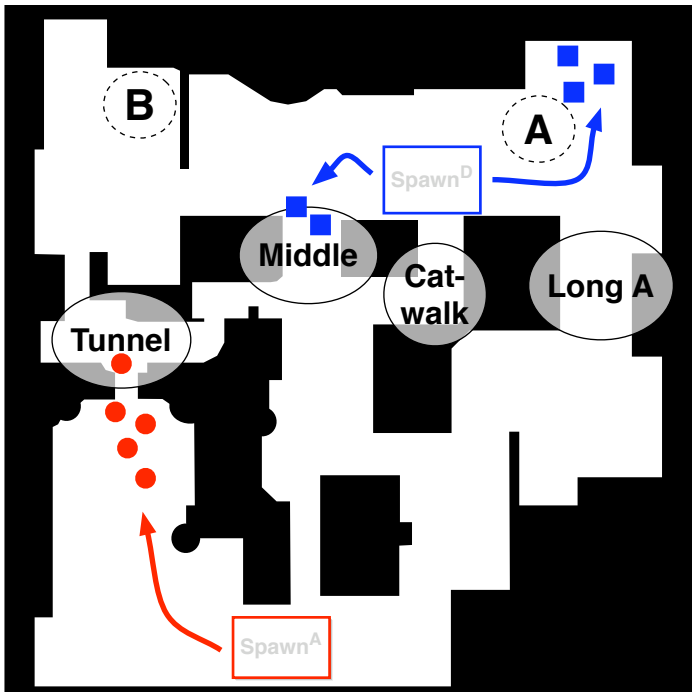


Figure 6.3: Illustration of the opening phase of a match in *CSS*'s popular map *dust2*. Attackers are represented by filled circles, defenders by filled squares. Here, a rush attack is launched via the tunnel, while the defense split their forces between the middle and site *A*.

6.1.2 Gameplay Example

The following example demonstrates important aspects of *CSS* gameplay during the initial stages of a match. In particular, this example shows how a team's strategic choices can translate into tactical advantages—or disadvantages. This example is divided into three stages: (1) an initial stage wherein players load in their spawn locations, (2) a stage during which each team executes an opening, and (3) a stage of tactical conflict.

Stage 1: Spawn and Freeze-time

When a match begins, the players of each team are randomly positioned in their respective spawn areas (Figure 6.2, right). In *dust2*, the attacker spawn is located in the map's southern region and the defender spawn is located in the map's northern region. After the players have been positioned, a specified 'freeze-time' elapses. During this time players can trade in-game currency for equipment upgrades, or formulate a strategy.

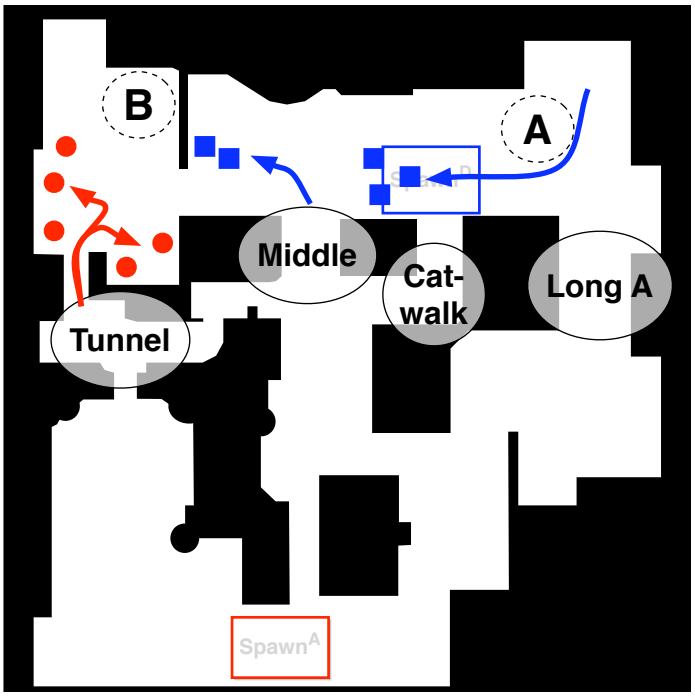


Figure 6.4: Illustration of the 'middlegame' of a match in *CSS*'s popular map *dust2*. Choices made in the opening phase can strongly influence the later game. Here, the attackers have developed a strong base near site *B*, putting the defending team at a tactical disadvantage.

Stage 2: The Opening Phase

When the freeze-time elapses, the game is unpaused and the players can move. Due to the distance between spawn locations in *dust2*, both teams act with limited information of their opponents' activities.[2] In Figure 6.3, all five attackers engage in a rush attack on site *B* through the tunnel. Meanwhile, the defenders have split into two groups: two defenders are sent to guard site *B*, while the remaining three move to watch the middle of the map.

Stage 3: Tactical Conflict

When the members of opposing teams come face-to-face, strategic choices made in the early game translate into clear tactical advantages. In particular, the attackers have developed positions near site *B* before an organised defense could be established (Figure 6.4).

[2]The *dust2* map features a line of sight that runs vertically from the attacker spawn up towards the top of the map; players can and occasionally do stop to attack each other from across the map.

6.1.3 Description of Data

The data consists of automatically sampled player locations[3] from two sources. The first is human gameplay (i.e., the only participants are human players).[4] The second source is bot gameplay, as carried out by Michael Booth's official *CSS* bots [2004],[5] (see Section 2.1.4). Obvious outliers have been removed from the human data (see Appendix B). The remaining data consists of 264 human matches and 252 bot matches which contain a total of 2640 and 2520 recorded trajectories respectively, split between attackers and defenders.

6.2 Categorical Analysis of Game Openings

This section describes exploratory data analysis of the trajectories that arise from human and bot behaviour in *CSS*. The objective is to analyse ways in which an *individual* player's behaviour can influence the outcome of a game, and ways in which humans and bots differ.

6.2.1 Experimental Setup

Opening book models for the attacking and defending teams, labelled $\bigcirc$ and $\square$ respectively, are constructed using prototypes derived from K-means clustering[6] with $K = 5$:

$$\bigcirc = \left\{ \ \bigcirc_0, \bigcirc_1, \bigcirc_2, \bigcirc_3, \bigcirc_4 \ \right\},$$
$$\square = \left\{ \ \square_0, \ \square_1, \ \square_2, \ \square_3, \ \square_4 \ \right\}.$$

The selection of $K = 5$ maintains adequate sample sizes across categories while capturing a variety of different trajectories that players follow. To create prototypes only out of uninterrupted openings, those openings in the data containing less than 10 points were withheld during model construction; the resulting set contains 175 human games (with 875 total trajectories per team) and 169 bot games (with 845 total trajectories per team).

After model construction, every trajectory in the data—including any trajectory containing fewer than 10 points—is mapped to its associated prototype, and the end-game outcome with which it is associated (win or loss) is tallied in a contingency table.

[3] The raw data contains additional information, including heading, weapon selection, etc. For a complete description, refer to Appendix B.

[4] Gameplay logs were collected from a large competitive tournament called Fragapalooza [Fragapalooza, 2008] by the Alberta Ingenuity Centre for Machine Learning (AICML) in July 2006 and 2007.

[5] I wish to extend my gratitude to Stephen Hladky for collecting this data.

[6] Prototypes imply a $\mathrm{categorise}$ function that discretises an opening space, as described in Section 4.

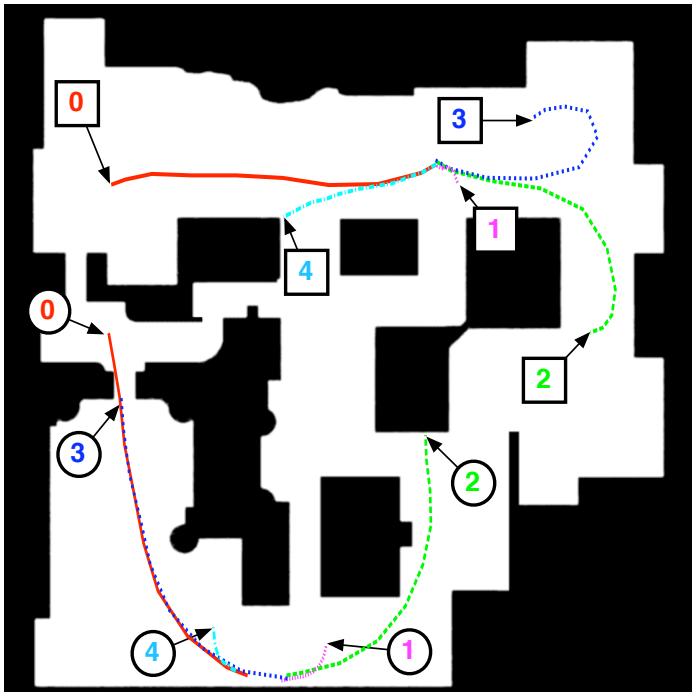


Figure 6.5: Illustration of prototype trajectories. Numbered circles indicate trajectories followed by attackers, originating from the bottom of the map. Numbered squares indicate trajectories followed by defenders, originating from the top of the map.

6.2.2 Results for Human Play

The results for human play reveal significant correlations between the trajectory an individual player chooses to follow and the outcome of a game. A labelled illustration of the resulting prototypes is presented in Figure 6.5, while the corresponding contingency tables with entries for each prototype are presented in Table 6.1.

The prototypes cover a variety of different opening trajectories. Some of the trajectories extend purposefully towards key strategic locations on the map (i.e., $\bigcirc_0$, $\bigcirc_3$, $\bigcirc_2$, and $\square_0$, $\square_2$, $\square_3$), while others remain close to spawn areas, possibly indicative of a late start.

$\bigcirc_i$	Win	Lose	Total	Win Rate
$\bigcirc_0$	167	116	283	0.59
$\bigcirc_1$	167	120	287	0.58
$\bigcirc_2$	213	196	409	0.52
$\bigcirc_3$	92	124	216	0.43
$\bigcirc_4$	56	69	125	0.45
Total	695	625	1320	0.53

$\square_i$	Win	Lose	Total	Win Rate
$\square_0$	159	141	300	0.53
$\square_1$	117	155	272	0.43
$\square_2$	116	121	237	0.49
$\square_3$	97	121	218	0.44
$\square_4$	131	162	293	0.45
Total	620	700	1320	0.47

Table 6.1: Contingency tables for human players. The contingency table for openings associated with the attacking and defending teams are shown on the left and right respectively.

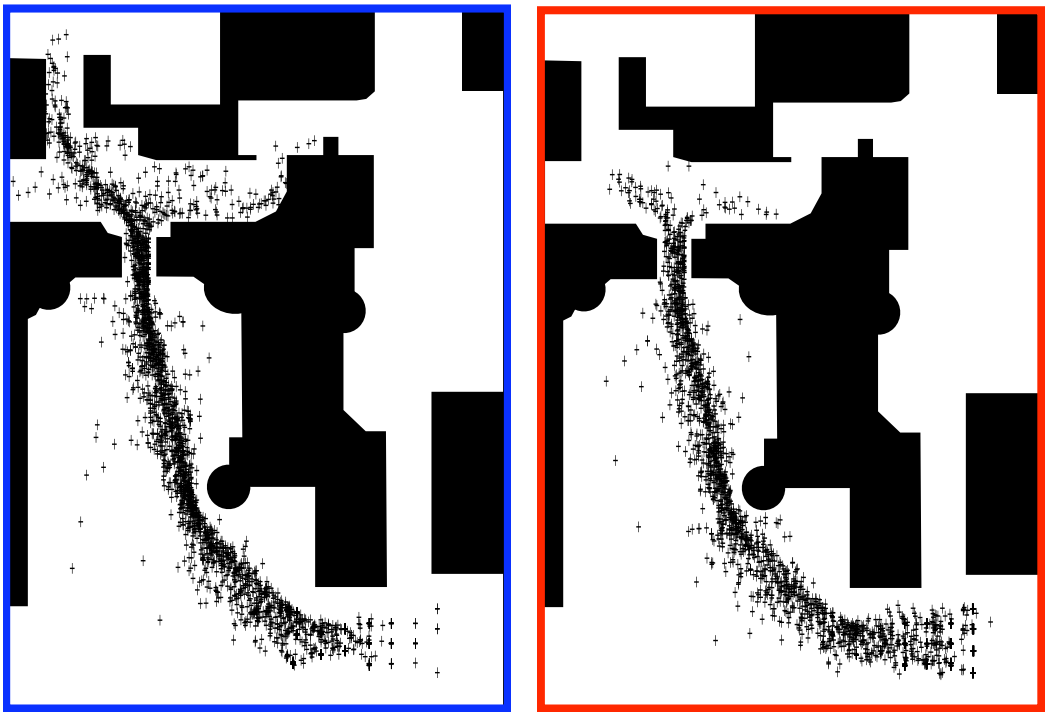


Figure 6.6: Visualisation of locations visited when human players follow $\bigcirc_0$ and $\bigcirc_3$, shown left and right respectively. $\bigcirc_0$ has higher visitation frequency towards the top of the cropped region, while $\bigcirc_3$ has higher visitation frequency towards the bottom-right.

The win rates are consistently close to average (i.e., there are no overwhelmingly 'good' trajectories to follow); this is an expected result, as *CSS* is a professional offering for which gameplay has been carefully engineered to be fair and balanced for both sides.

One striking result is how prototypes $\bigcirc_0$ and $\bigcirc_3$ produce significantly different[7] outcomes despite resembling one-another. Close inspection of the point locations at which these trajectories begin (i.e., the locations within which players randomly spawn), as shown in Figure 6.6, reveals that players who take one of $\bigcirc_0$ or $\bigcirc_3$ and go on to *win* are significantly[8] more likely to have spawned further *left* than losing players. The location in which a player randomly spawns may impact his ability to succeed with a particular opening.

Another striking result concerns the defender prototypes. Specifically, there is a significant difference[9] between game outcomes resulting from $\square_0$ and $\square_4$, which are both characterised by leftward movement. This may be because attackers tend to travel along the extreme left and right sides of the map; defenders enacting $\square_4$ may engage fewer attackers.

[7] Fisher's exact test for independence gives a two-tailed P value of 0.0003.
[8] An unpaired t-test of the trajectories' initial x coordinates yields a two-tailed P value of 0.0170.
[9] Fisher's exact test for independence gives a two-tailed P value of 0.0487.

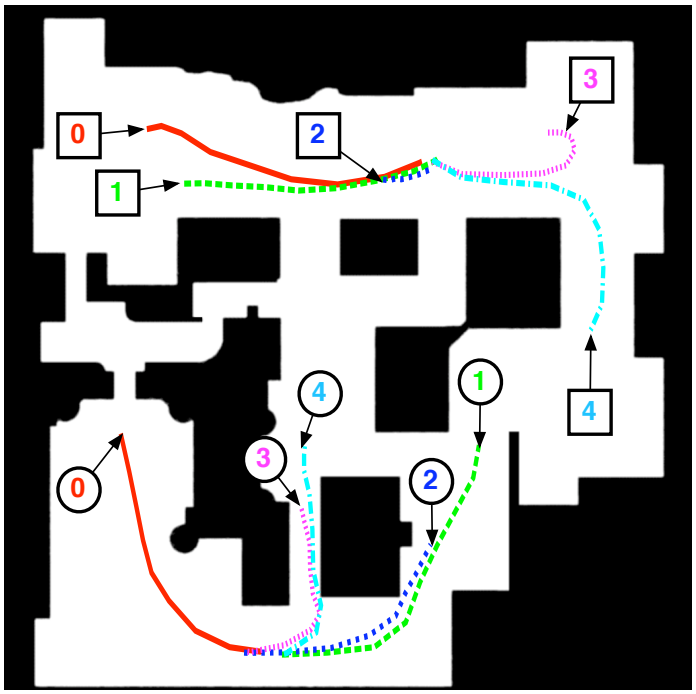


Figure 6.7: Illustration of prototype trajectories derived from bot data. Prototypes indicated with numbered circles are derived from members of the attacking team, while those indicated with numbered squares are derived from members of the defending team.

6.2.3 Results for Computer Play

The results for computer play, illustrated in Figure 6.7 and tabulated in Table 6.2, reveal fewer unexpected correlations between individual agent behaviour and game outcomes. However, Figure 6.7 highlights key differences between computer play and human play.

First, the bot defense strategies are shown to closely resemble human defense strategies, while bot attack strategies involve significantly more travel through middle of the map (in particular, bot strategies $\bigcirc_3$ and $\bigcirc_4$ constitute almost *half* the data for bots).

$\bigcirc_i$	Win	Lose	Total	Win Rate
$\bigcirc_0$	93	96	189	0.49
$\bigcirc_1$	82	64	146	0.56
$\bigcirc_2$	74	75	149	0.50
$\bigcirc_3$	193	191	384	0.50
$\bigcirc_4$	208	184	392	0.53
Total	650	610	1260	0.52

$\square_i$	Win	Lose	Total	Win Rate
$\square_0$	103	122	225	0.46
$\square_1$	180	194	374	0.48
$\square_2$	72	75	147	0.49
$\square_3$	143	166	309	0.46
$\square_4$	112	93	205	0.54
Total	610	650	1260	0.48

Table 6.2: Contingency tables for bot players. The contingency table for openings associated with the attacking and defending teams are shown on the left and right respectively.

Model	Parameter Range	Cross-validation	Other
Opening Book	$1 \leq K \leq 50$	10-fold	K-means derivation
Opening Book	$1 \leq K \leq 50$	10-fold	K-medoids derivation
Markov Model	$1 \leq m \leq 50$	10-fold	$\kappa(t) = \sqrt{1 + 500mt}$
k-NN	$1 \leq k \leq 50$	10-fold	$\alpha(T) = \text{dist}(T, Q)^{-1}$

Table 6.3: Experimental settings for models applied to the prediction task.

Additionally, some of the defense strategies that appear analogous between the two groups (e.g., *human* $\square_0$ and *bot* $\square_0$; *human* $\square_2$ and *bot* $\square_4$) have disparate win rates, although the differences are not statistically significant. Testing with a larger dataset may potentially reveal that openings that work well for humans do not necessarily work well for bots—an observation Hyatt makes of human and computer chess players [1999].

The trajectories analysed in this section do not take into account the actions of opponents or teammates.[10] Despite this, the results presented in this section demonstrate that cluster analysis can be useful in revealing significant correlations between an individual player's actions and his or her team's chances for success. However, it remains to be shown whether these prototypes retain an accurate representation of the trajectories that actually occur in play. The proceeding sections, wherein prototyping is compared to other modelling approaches in tasks of prediction and classification, are designed to address this question.

6.3 Empirical Results on the Prediction Task

The prediction task is a test of a model's ability to estimate a *complete* trajectory from its partial execution (see Section 4.2). The following sections detail the experimental setup and results attained by each model on human trajectory data over a range of parameters.

6.3.1 Experimental Setup

Experiments are run over four models of trajectory data. Their settings are tabulated in Table 6.3. In addition, to test a model's ability to predict every trajectory in the dataset, ten-fold cross-validation is employed (see Appendix A.3) for each model.

Two prototype models are constructed (Section 5.1), one using K-means to derive prototypes and one using K-medoids to derive prototypes. For each, the parameter K (indicat-

[10] Opposing players often *do* interact in the opening phases of play in *dust2* (refer to the footnote on page 50), and teammates regularly influence one-another's behaviour when enacting cooperative strategies.

ing number of prototypes) is explored from 1 to 50 in increments of 1. A Markov model of trajectories (Section 5.2) is explored with its state-size parameter m ranging from 1 to 50, also in increments of 1 (this m affects the number of Markov states at time t according to the function $\kappa(t) = \sqrt{1 + 500mt}$). Nearest neighbour estimation (Section 5.3) is similarly explored for neighbourhoods of size 1 to neighbourhoods of size 50 in increments of 1; the trajectories in each neighbourhood are uniformly weighted (i.e., $\alpha(T) = \text{dist}(T, Q)^{-1}$).

6.3.2 Results

The Opening Book approach using K-means derived prototype trajectories quickly settles on a prediction error of around 800 (Figure 6.8, top-left); the same approach using K-medoids derived prototypes has similar results (Figure 6.8, top-right), maintaining a relatively steady error rate of around 775 including and after $K = 10$.[11] In general, derivation using either technique (K-means or K-medoids) appears to yield similar results.

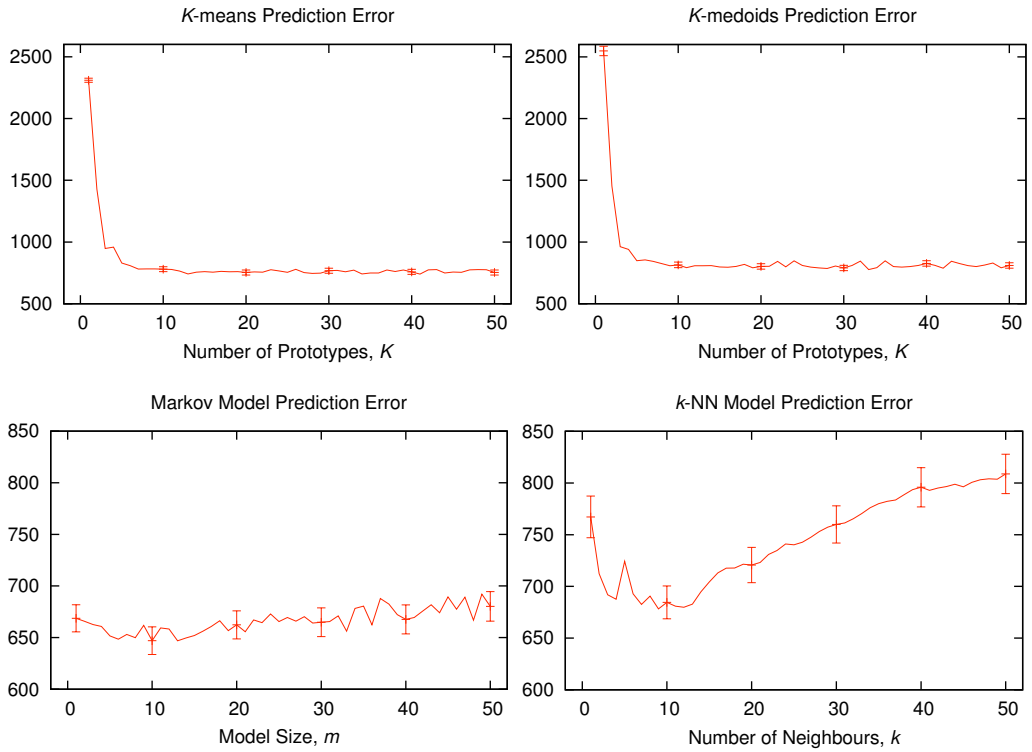


Figure 6.8: Prediction error for each of the four models. K-means and K-medoids are shown top-left and top-right respectively; the Markov model and nearest neighbour estimation are shown bottom-left and bottom-right respectively. Error bars show standard error.

[11] As K increases to the size of the dataset, any prototype model will eventually be equivalent to nearest neighbour estimation with $k = 1$, which, as illustrated in Figure 6.8, produces results close to 775 as well.

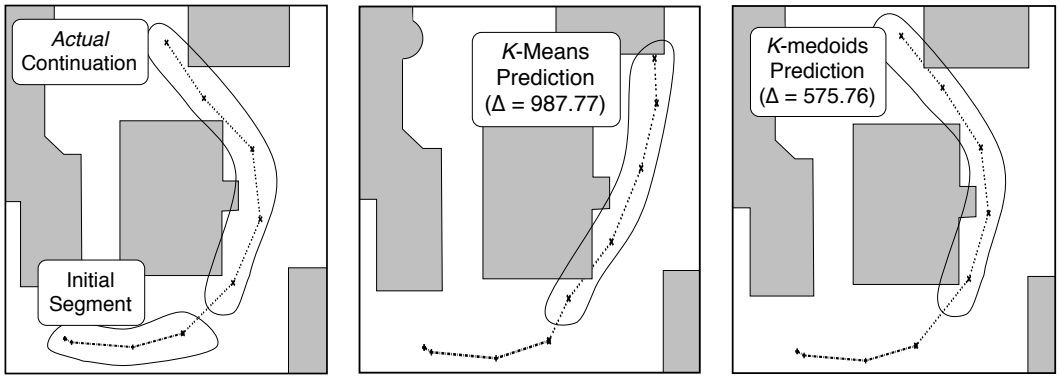


Figure 6.9: Qualitative examples of predicted continuations generated by a K-means model with $K = 41$ and a K-medoids model with $K = 33$. These predictions differ from the actual continuation (shown left) by the amount indicated by the variable Δ.

The Markov model and nearest neighbour estimation produce superior prediction accuracy to the prototype models. The Markov model attains a best prediction accuracy of 646.89 at $m = 13$—the best of any of the four models (Figure 6.8, bottom-left). Nearest neighbour estimation attains a best prediction accuracy of 678.27 at $k = 9$, after which the prediction error increases with increasing values of k (Figure 6.8, bottom-right).

Qualitatively, each model tends to produce smooth trajectories that resemble actual character movement within *CSS*. To provide intuitive examples of the relationship between trajectories and the distances between them, Figures 6.9 and 6.10 illustrate predictions from various paramaterisations of the models on an example initial segment.

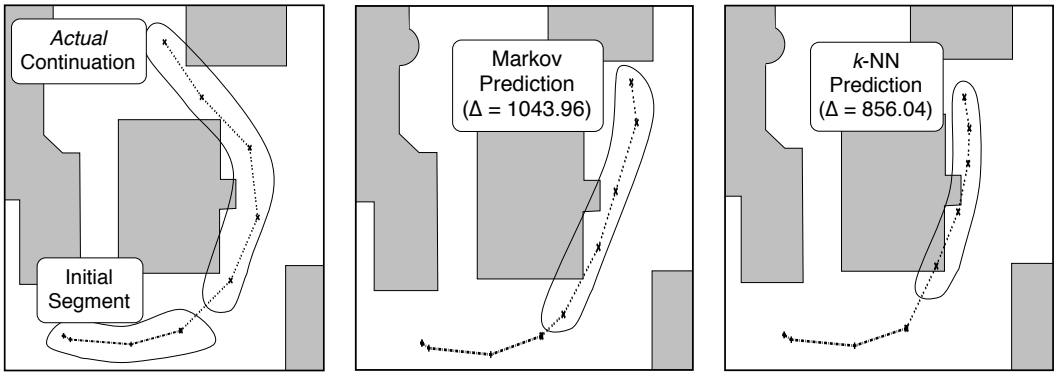


Figure 6.10: Qualitative examples of predicted continuations produced by the Markov model with $m = 13$ and nearest neighbour estimation with $k = 41$. These predictions differ from the actual continuation (shown left) by the amount indicated by the variable Δ.

Model	Parameter Range	Cross-Validation	Other
Opening book	$1 \leq K \leq 50$	10-fold	K-means derivation
Opening book	$1 \leq K \leq 50$	10-fold	K-medoids derivation
Markov model	$1 \leq m \leq 50$	10-fold	$\kappa(t) = \sqrt{1 + 500mt}$
k-NN	$1 \leq k \leq 50$	10-fold	$\alpha(T) = \text{dist}(T, Q)^{-1}$

Table 6.4: Experimental settings for models applied to the classification task.

6.4 Empirical Results on the Classification Task

The classification task measures a model's ability to help discern the trajectory data produced by two different groups (see Section 4.3). The following sections describe the experimental setup and present results comparing the empirical performance for each model.

6.4.1 Experimental Setup

The experimental settings, tabulated in Table 6.4, are the same as in the prediction task (Section 6.3.1). Specifically, paramaters for each model are explored from 1 to 50, and ten-fold cross-validation ensures that each model is tested on each trajectory at least once.

6.4.2 Results

Classification accuracy attained by the 'Opening Book' models, consisting of either K-means or K-medoids derived prototypes, is shown to improve as K increases (Figure 6.11, top). The performance of these two models is revealed to plateau early on with classification accuracy just below 90%; the curvature of the plots suggest diminishing returns for increasingly large values of K. The sample confusion matrices presented in Table 6.5, top, reveal no strong biases in the models towards correctly identifying bot or human play.

K-Means Opening Book

Class	Correct	Incorrect	Accuracy
Human	767	108	87.7%
Bot	775	70	91.2%

K-Medoids Opening Book

Class	Correct	Incorrect	Accuracy
Human	778	97	88.9%
Bot	726	119	85.2%

Markov Model

Class	Correct	Incorrect	Accuracy
Human	605	270	69.1%
Bot	771	74	91.2%

Nearest Neighbour Estimation

Class	Correct	Incorrect	Accuracy
Human	875	72	92.2%
Bot	826	19	97.8%

Table 6.5: Confusion matrices for K-means and K-medoids derived with $K = 50$ and 35 respectively, for the Markov Model with $m = 40$, and for nearest neighbours with $k = 1$.

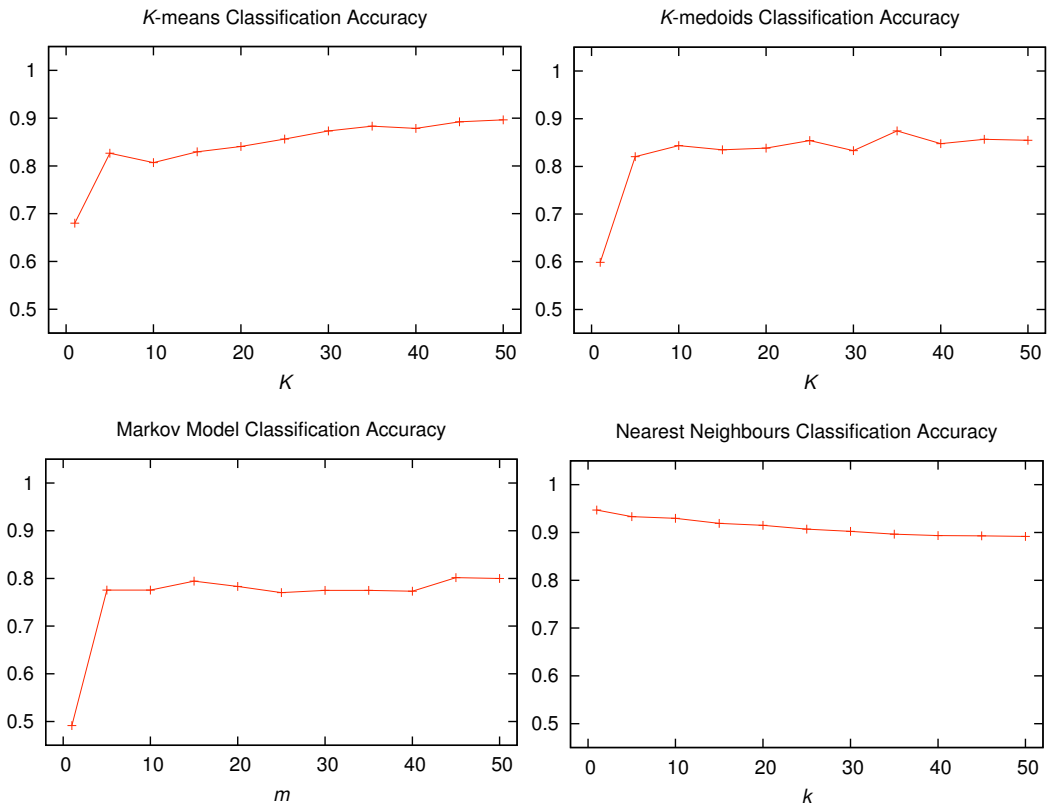


Figure 6.11: Classification accuracy for each model. K-means and K-medoids derivation of prototypes are shown top-left and top-right, respectively; the Markov model and nearest neighbour estimation results are shown bottom-left and bottom-right respectively.

The Markov model of trajectories has the least reliable classification accuracy (Figure 6.11, bottom-left). A sample confusion matrix (Table 6.5, bottom-left) reveals that this inaccuracy is primarily due to the model's inability to correctly identify human play. When the parameter m exceeds 5, the model's classification accuracy plateaus significantly. This weak result is contrary to the Markov model's superior performance on the prediction task.

Nearest neighbour estimation has the best classification accuracy of any of the models (Figure 6.11, bottom-right). This high level of performance may have been achieved due to the regularity, or lack of noise, with which bots generate trajectories compared to humans.[12] Classification accuracy deteriorates with increasing values of k, which runs contrary to the earlier result that larger neighbourhoods can benefit *prediction* accuracy (Figure 6.8). The confusion matrix for this approach (Table 6.5, bottom-right) reveals that the model reliably identifies both bot and human play, with a slight bias towards correctly identifying bot play.

[12]The regularity of bot-generated trajectories compared to human-generated trajectories is supported by Figure 4.4 on page 31, which shows that bots favor specific channels of travel more than human players do.

6.5 Summary

This chapter detailed the results of three empirical studies. The first demonstrated that a selective discretisation of an opening space (here, prototype modelling by means of K-means cluster analysis) can help to uncover significant correlations between individual player actions and game outcomes in tactical shooters. The second and third experiments explored such a model's accuracy in comparison to baselines. The nearest neighbour approach maintained a representation that achieved better classification results, while the Markov model demonstrated superior performance on trajectory prediction.

Chapter 7

Discussion

This chapter provides an overview of the limitations and potential future directions for the work presented in this study. The limitations that will be described are largely data-centric and focus on the dataset, how it was used, and the nature of the information that could be discovered in it. Indications for future work from this research are various: the consideration of additional distance metrics could prove beneficial, and there are potential practical applications for the extensions of the techniques described in this study.

7.1 Limitations and Problems Encountered

Three primary limitations can be identified in this study. The first concerns the size of the dataset that underwent analysis. The second is due to simplifying assumptions that were made about *CSS* gameplay. And the third lies in the an inability to recognise the existence of a *causal* relationship between openings and game outcomes.

Limitation 1: Dataset

There exists no formal mechanism with which to verify the quality of the gameplay data used in this study, but there are strong supporting arguments in its favor. In particular, Fragapalooza [2008] attendees are typically skilled players whose goal is to win the matches they play; additionally, the games analysed in this study were played over a low-latency local network connection. The *quantity* of gameplay examples is more concerning.

The relatively small size of the dataset precludes analysis of the joint strategies enacted by a *team* of players, let alone the interplay between two competing teams. For an idea

of the data requirements necessary to perform this analysis, consider an over-simplified opening book consisting of just 2 prototype trajectories. With 2 prototypes for each player to choose from, a team of 5 identical players can act jointly in one of 6 unique ways. Two of these teams competing against each other would be able to interact in $6 \times 6 = 36$ unique ways. The results of the empirical study in Section 6.2 suggest that sample sizes in excess of 100 games are necessary to make statistically significant comparisons between individual player choices in *Counter-Strike: Source*. An adequate sample of games in this simple setting might exceed $3,600$, significantly more than the current 264 in the dataset.

Limitation 2: Missing Gameplay Elements

By considering only *spatial* trajectories, I disregard several time-variable elements that may be crucial to *CSS* gameplay. These elements include movement control variables and the effects of weapons, equipment, and past rounds of play on the trajectories players follow.

A number of variables contribute to a player's maximum travel speed (i.e., the distance between the points along the trajectories he or she produces). Players can increase their character's maximum travel speed by having their character holster its primary weapon, a riskier alternative to travelling with a weapon readied.[1] Players can also make tradeoffs between speed and stealth.[2] While it is true that the spatial trajectories analysed in this study imply speed by the distance between measurements, they omit the combination of variables employed by a player to achieve that particular speed of travel.

A number of tactical variables can influence the trajectories players choose. A character's equipment and weapon selection can dictate appropriate tactics, thereby influencing the trajectory the controlling player chooses. Additionally, players can conceal their character's movement by covering an area with smoke or by blinding opponents,[3] and whether these specific tactics are successful or not can strongly influence a player's future trajectory. Finally, winners in *CSS* are determined across *multiple* rounds of play—it is common for teams to choose their strategies conditioned on the events of previous rounds of play.

[1] In *CSS*, players can choose to wield a weaker auxiliary weapon while carrying their primary weapon on their shoulder. Doing so enables players to travel quickly at the risk of being caught offguard.

[2] Players can choose between crawling (which is silent and presents a smaller target, but slow), walking (which is silent but slow), and running (which generates more noise, but enables a player to travel quickly).

[3] In *CSS* the *smoke grenade* creates a visibility barrier that players can travel behind unseen. The *flashbang* grenade blinds players (both teammates and opponents) whose view frustums are directed at it.

Limitation 3: Identifying a Causal Relationship

The analysis in Section 6.2 revealed correlations between game outcomes and the trajectories followed by an individual player; whether this relationship is causal is difficult to discern for two reasons. First, a player's spatial location may implicitly contain information about the state of the game in which it occurs.[4] For example, a trajectory leading deep into enemy territory may only be possible when the area is poorly defended. Second, the trajectory a player follows may be due to his/her expertise; expert players may be more likely to win due to their expertise and not necessarily the trajectories they tend to choose.

7.2 Future Work

This section describes future areas of research related to this study. I consider distance metrics beyond simple straight-line distance that might provide a better approximation to the distance between points in an environment, other groups among which classification based on trajectory data can be applied, and the implications of this study for the designers of computer characters in tactical shooters and combat simulations in general.

Distance Measures

This study employed Euclidean distance to measure distance between points and trajectories (Section 3.2). However, in *CSS* and other tactical shooters, in which there are obstacles, the 'travel' distance between two points can be strongly influenced by the environment. It is possible that travel distance could be derived from data, by observing the routes players take when travelling between points in different situations. It is also worth noting that Euclidean distance is equally sensitive to distance along all axes, but in determining cluster membership, a distance metric that is variably sensitive along its axes—such as Mahalanobis distance [1936]—can often be desirable. Intuitively, Mahalanobis distance is a metric for which the locus of points considered equidistant from a centre may be ellipsoidal. The distance from a cluster of trajectories with low variance among certain segments (e.g., trajectories with a common source or bottleneck) may be appropriately measured in this way.

[4] I have attempted to mitigate this risk by cropping trajectories at an early cutoff of 10 points, but because of the rich input *CSS* players receive (which includes aural as well as visual input) I cannot be certain that the points along a trajectory do not imply more than mere location.

Additional Classification Tasks

There are numerous practical applications of techniques that can reliably classify play. In this study, I explored the use of trajectory models to discern whether a trajectory was human-generated or computer-generated. These results concerned two very different groups that access the game using different interfaces, but the encouraging results suggest that it may be possible to use precisely the same methods—without introducing any change to the algorithms—to classify among other categories. For example, the ability to classify among levels of *expertise*, based on in-game behaviour, has the potential to complement existing online ranking systems which pit players against equally matched opponents. The ability to classify among cheaters and honest players has the potential to complement existing anti-cheating systems—for example, players who use *wallhacks* (tools that allow players to see through walls) exhibit idiosyncratic behaviours that include staring into solid walls or reacting to opponents before they are visible [Carless, 2004]. Games like Turn 10 Studios's *Forza Motorsport 2* [2007] feature systems that model a player's behaviour for entertainment purposes [Navarro, 2007]; these techniques may prove useful in similar endeavours.

Improving Bots

Despite its rich interactive experience, I have shown that certain elements of *CSS* gameplay remain predictable (Section 6.3). It is interesting to consider this alongside the informed opinion of Steven Polge, the designer behind the computer-controlled players in the popular *Unreal* video game franchise [Baker, 2002]:

> *The definition of "indistinguishable from human opponents" is a moving target, as games evolve to provide a richer interactive experience. It's not hard to make an AI indistinguishable from a human for a simple game like Tic Tac Toe. As the game becomes more complex, it gets harder.*

Trajectory models similar to those used in this study can potentially assist bot decision making during gameplay phases in which characters follow predictable routines.

7.3 Summary

This chapter described inherent limitations and future directions for my study. The majority of the limitations indicate a shortage of player data. Additional data would enable the analysis of important gameplay elements that are not captured by spatial trajectories, including weapon use, skill level, and interplay between teams. I also stressed that, while having uncovered statistically significant correlations between opening trajectories and game outcomes, this study has not identified a causal relationship between the two. Finally, I described some potential areas of application and future research for the techniques explored in this thesis, including the use of alternative distance measures, alternative classification tasks, and improving bot behaviour by providing a better approximation to human play.

Chapter 8

Conclusions

In the communities surrounding tactical shooters, the widespread use of natural language to describe openings causes ambiguity. Indeed, the evidence in support of any one opening is necessarily anecdotal because there is no algorithmic way to recognise it. This problem exists because these openings are not formally defined.

I presented a survey of related work on tactical behaviour in games, believability testing, the computational analysis of openings, and modelling agent motion. This survey suggests that no prior work addresses the problem of formally defining openings in tactical gameplay.

Towards solving this problem, I proposed *deriving* these definitions by clustering player-generated trajectory data. To provide an objective measure of the quality of these definitions, I proposed two empirical tasks: a prediction task wherein future points along a trajectory are predicted from that trajectory's initial segment, and a classification task wherein human trajectories are discerned from bot trajectories. Empirical results show that:

1. Cluster analysis over player-generated trajectory data reveals statistically significant correlations between a player's actions and the outcome of a game.

2. Compared to baseline models, these definitions accurately model player motion in that they can be used to predict and classify player-generated trajectory data.

This approach has exciting potential practical application to many related platforms, as well as to player classification and the control of computer-controlled opponents and allies.

Bibliography

[Aaltonen, 2005] Christian Aaltonenn Aaltonen. Counter-Strike: Source Strategy Guide. Accessed June 30 2008, http://aaltonen.us/2005/01/12/counter-strike-source-strategy-guide/, January 2005.

[Adomavicius and Tuzhilin, 2005] Gediminas Adomavicius and Alexander Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[Aha, 1990] David W. Aha. *A Study of Instance-Based Algorithms for Supervised Learning Tasks*. PhD thesis, University of California, Irvine, 1990.

[Allis, 1988] Victor Allis. A Knowledge-Based Approach to Connect-Four - The Game is Solved: White Wins. Master's thesis, Vrije Universiteit, Amsterdam, 1988.

[Arthur and Vassilvitskii, 2006] David Arthur and Sergei Vassilvitskii. How Slow is the k-Means Method? In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SCG-06)*, pages 144–153, New York, NY, USA, 2006. ACM.

[Baker, 2002] Tracy Baker. Game Intelligence: AI Plays Along. *Computer Power User*, 2(1):56–60, January 2002.

[Bakkes *et al.*, 2004] Sander Bakkes, Peter Spronck, and Eric Postma. TEAM: The Team-oriented Evolutionary Adaptability Mechanism. In *Entertainment Computing - ICEC 2004, Third International Conference*, pages 273–282, Endhoven, the Netherlands, September 2004.

[Bakkes *et al.*, 2005] Sander Bakkes, Pieter Spronck, and Eric Postma. Best-Response Learning of Team Behaviour in Quake III. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

[Barlow and Morrison, 2005] Michael Barlow and Peter Morrison. Challenging the Super Soldier Syndrome in 1st Person Simulations. In *Proceedings of SimTecT 2005 Conference*, Sydney, Australia, May 2005.

[Bererton, 2004] Curt Bererton. *Multi-Robot Coordination and Competition Using Mixed Integer and Linear Programs*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, August 2004.

[Bermejo and Cabestany, 2001] Sergio Bermejo and Joan Cabestany. Oriented Principal Component Analysis for Large Margin Classifiers. *Neural Networks*, 14(10):1447–1461, December 2001.

[Billings *et al.*, 2003] Darse Billings, Neil Burch, Aaron Davidson, Robert C. Holte, and Jonathan Schaeffer. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 661–668, Acapulco, Mexico, August 2003.

[Bohemia Interactive Studio, 2001a] Bohemia Interactive Studio. Operation Flashpoint: Cold War Crisis. Codemasters, http://www.flashpoint1985.com/, August 2001.

[Bohemia Interactive Studio, 2001b] Bohemia Interactive Studio. Operation Flashpoint Photos from Action. Accessed June 30 2008, http://www.flashpoint1985.com/photos/photos.html, 2001.

[Booth, 2004] Michael Booth. The Official Counter-Strike Bot. In *Proceedings of the Game Developers' Conference (GDC-04)*, 2004.

[Buro, 1999] Michael Buro. Toward Opening Book Learning. *International Computer Chess Association (ICCA) Journal*, 22(2):98–102, 1999.

[Campbell *et al.*, 2002] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[Campbell, 1999] Murray Campbell. Knowledge Discovery in Deep Blue. *Communications of the ACM*, 42(11):65–67, 1999.

[Carless, 2004] Simon Carless. *Gaming Hacks*. O'Reilly Media, Inc., 2004.

[Chandrasekaran *et al.*, 2002] Balakrishnan Chandrasekaran, John R. Josephson, Bonny Banerjee, Unmesh Kurup, and Robert Winkler. Diagrammatic Reasoning in Support of Situation Understanding and Planning. In *Proceedings of the 23rd Army Science Conference*, 2002.

[Chang *et al.*, 2002] Francis Chang, Wu-chang Feng, Wu-chi Feng, and Jonathan Walpole. Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 151–156, Marseille, France, 2002. ACM.

[Charness, 1991] Neil Charness. Knowledge and Search in Chess. *Towards a General Theory of Expertise*, pages 39–63, 1991.

[Cole *et al.*, 2004] Nicholas Cole, Sushil J. Louis, and Chris Miles. Using a Genetic Algorithm to Tune First-Person Shooter Bots. In *Proceedings of the International Congress on Evolutionary Computing*, pages 139–145, Portland, Oregon, 2004. IEEE Press.

[Darken *et al.*, 2004] Christian Darken, David Morgan, and Gregory Paull. Efficient and Dynamic Response to Fire. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, 2004.

[Darken, 2007] Christian Darken. Level Annotation and Test by Autonomous Exploration. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-07)*, Stanford, California, 2007.

[Dasarathy, 1990] Belur V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, 1990.

[Destineer, 2005a] Destineer. Close Combat: First to Fight. 2K Games and Macsoft, http://xbox.ign.com/objects/673/673931.html, April 2005.

[Destineer, 2005b] Destineer. Close Combat: First to Fight Combined Arms Video. Accessed June 30 2008, http://hlfallout.filecloud.com/files/file.php?file_id=1247, 2005.

[Doyle, 1999] Patrick Doyle. Virtual Intelligence from Artificial Reality: Building Stupid Agents in Smart Environments. In *Working Notes of the 1999 AAAI Spring Symposium on Artificial Intelligence and Computer Games*, 1999.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. Semantics for Hierarchical Task-Network Planning. Technical report, University of Maryland at College Park, College Park, MD, USA, 1994.

[Fikes and Nilsson, 1971] Richard Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving by Problem Solving. In William Kaufmann, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI-71)*, pages 608–620, 1971.

[Fragapalooza, 2008] Fragapalooza. http://fragapalooza.com, 2008.

[Geisler, 2002] Benjamin Geisler. An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a 'First Person Shooter' Video Game. Master's thesis, Department of Computer Sciences, University of Wisconsin-Madison, 2002.

[Gorman *et al.*, 2006a] Bernard Gorman, Christian Thurau, Christian Bauckhage, and Mark Humphrys. Bayesian Imitation of Human Behavior in Interactive Computer Games. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR 06)*, pages 1244–1247, Washington, DC, USA, 2006. IEEE Computer Society.

[Gorman *et al.*, 2006b] Bernard Gorman, Christian Thurau, Christian Bauckhage, and Mark Humphrys. Believability Testing and Bayesian Imitation in Interactive Computer Games. In *Simulation of Adaptive Behaviour (SAB-06)*, pages 655–666, Rome, Italy, September 2006.

[Grajkowski, 2006] Jeffery Grajkowski. Unofficial Definition of CSViz Logfile Format. Accessed June 30 2008, http://www.cs.ualberta.ca/ csviz/technical/logformat.txt, 2006.

[Graphpad Software, Inc., 2005] Graphpad Software, Inc. Graphpad QuickCalcs: Free Statistical Calculators. Accessed June 30 2008, http://www.graphpad.com/quickcalcs/, 2005.

[Han and Kamber, 2006] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques, 2nd ed.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, March 2006.

[Hastie *et al.*, 2001] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[Hayward *et al.*, 2005] Ryan Hayward, Yngvi Björnsson, Michael Johanson, Morgan Kan, Nathan Po, and Jack van Rijswijck. Solving 7×7 Hex with Domination, Fill-In, and Virtual Connections. *Theoretical Computer Science*, 349(2):123–139, 2005.

[Hoang *et al.*, 2005] Hai Hoang, Stephen Lee-Urban, and Hector Muñoz-Avila. Hierarchical Plan Representations for Encoding Strategic Game AI. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)*. AAAI Press, 2005.

[Hyatt, 1999] Robert M. Hyatt. Book Learning: A Methodology to Tune an Opening Book Automatically. *International Computer Chess Association (ICCA) Journal*, 22(1):3–12, 1999.

[id Software, 1997] id Software. Quake II. http://www.idsoftware.com/games/quake/quake2/, December 1997.

[id Software, 1999] id Software. Quake III. http://www.idsoftware.com/games/quake/quake3-arena/, December 1999.

[Invision Gaming, 2007] Invision Gaming. Counter-Strike: Source Tickrate. Accessed June 30 2008, http://www.invision-gaming.co.uk/Counter-Strike-Source/CSS-Tickrate-61.html, 2007.

[Kaufman and Rousseeuw, 1990] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[Keogh and Kasetty, 2002] 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.

[Khoo and Zubek, 2002] Aaron Khoo and Robert Zubek. Applying Inexpensive AI Techniques to Computer Games. *IEEE Intelligent Systems*, 17(4):48–53, 2002.

[Kohavi, 1995] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1137–1145, 1995.

[Laird and Duchi, 2000] 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.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Laird, 2001] John E. Laird. It Knows What You're Going To Do: Adding Anticipation to a Quakebot. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Canada, 2001. ACM Press.

[Lee *et al.*, 2007] 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.

[Lewis and Barlow, 2005] Edward Lewis and Michael Barlow. The Use of Games to Investigate Tactical Decision-Making. In *Proceedings of SimTecT 2005 Conference*, Sydney, Australia, May 2005.

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

[Lidén, 2001] Lars Lidén. Using Nodes to Develop Strategies for Combat with Multiple Enemies. In *Artificial Intelligence and Interactive Entertainment: Papers from the 2001 AAAI Symposium*, pages 59–63, 2001.

[Lidén, 2002] Lars Lidén. Strategic and Tactical Reasoning with Waypoints. In *AI Game Programming Wisdom*. Charles River Media, 2002.

[Lidén, 2003] Lars Lidén. Artificial Stupidity: The Art of Intentional Mistakes. In *AI Game Programming Wisdom II*. Charles River Media, 2003.

[Lincke, 2000] Thomas R. Lincke. Strategies for the Automatic Construction of Opening Books. In Tony Marsland and Ian Frank, editors, *Proceedings of the Second International Conference on Computers and Games, Lecture Notes in Artificial Intelligence*, pages 74–86. Springer-Verlag, 2000.

[Livingstone and McGlinchey, 2004] Daniel Livingstone and Stephen J. McGlinchey. What Believability Testing Can Tell Us. In *Proceedings of the Conference on Game AI, Design and Education (CGAIDE-04)*, November 2004.

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

[MacQueen, 1967] James B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[Mahalanobis, 1936] Prasanta C. Mahalanobis. On the Generalized Distance in Statistics. In *Proceedings of the National Institute of Science (India)*, volume 12, pages 49–55, Calcutta, India, 1936.

[Manninen, 2001] Tony Manninen. Virtual Team Interactions in Networked Multimedia Games - Case: "Counter-Strike" - Multi-player 3D Action Game. In *The 4th Annual International Workshop on Presence (PRESENCE-01)*, Philadelphia, Pennsylvania, 2001.

[Markov, 1971] Andrey Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. In Ronald Howard, editor, *Dynamic Probabilistic Systems (Volume I: Markov Models)*, pages 552–577. John Wiley & Sons, Inc., New York City, 1971.

[Michie *et al.*, 1994] Donald Michie, David J. Spiegelhalter, and Charles C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA, February 1994.

[Monolith Productions, 2005] Monolith Productions. F.E.A.R.: First Encounter Assault Recon. Vivendi Universal, http://whatisfear.com/, 2005.

[Navarro, 2007] Alex Navarro. Forza Motorsport 2 for Xbox 360 Review. Accessed June 30 2008, http://www.gamespot.com/xbox360/driving/forzamotorsport2/review.html, May 2007.

[Nielsen Media Research, 2007] Nielsen Media Research. Playstation 2 Accounted for 42 Percent of Video Game Play in June. Accessed June 30 2008, http://www.nielsen.com/media/2007/pr_070726.html, 2007.

[Orkin, 2004] Jeff Orkin. Symbolic Representation of Game World State: Toward Real-Time Planning in Games. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, 2004.

[Orkin, 2006] Jeff Orkin. Three States and a Plan: The A.I. of F.E.A.R. In *Proceedings of the Game Developers' Conference (GDC-06)*, San Jose, California, March 2006.

[Paull and Darken, 2004] Gregory H. Paull and Christian J. Darken. Integrated On- and Off-Line Cover Finding and Exploitation. In *Proceedings of GAME-ON 2004*, 2004.

[Ratliff *et al.*, 2006] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum Margin Planning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML-06)*, pages 729–736, New York, NY, USA, 2006. ACM Press.

[Raven Software, 2002] Raven Software. Soldier of Fortune II. Activision, http://www.soldier-of-fortune.com/, May 2002.

[Reike and Boon, 2008] Zeid Reike and Michael Boon. Postmortem: Infinity Ward's Call of Duty 4. *Game Developer*, March 2008.

[Schaeffer *et al.*, 2007] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is Solved. *Science*, pages 1144079+, July 2007.

[Selby, 1999] Alex Selby. Optimal Heads-up Preflop Poker. Accessed June 30 2008, http://www.archduke.demon.co.uk/simplex/, 1999.

[Smith *et al.*, 2007] Megan Smith, Stephen Lee-Urban, and Hector Muñoz-Avila. RETALIATE: Learning Winning Policies in First-Person Shooter Games. In *Proceedings of the Nineteenth Innovative Applications of Artificial Intelligence Conference (IAAI-07)*, pages 1801–1806, 2007.

[Southey *et al.*, 2007] Finnegan Southey, Wesley Loh, and Dana Wilkinson. Inferring Complex Agent Motions from Partial Trajectory Observations. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[Spronck, 2005] Peter Spronck. A Model for Reliable Adaptive Game Intelligence. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

[Tamte, 2004] Peter Tamte. GameSpy: Close Combat: First to Fight - Vol #1 (PC). Accessed June 30 2008, http://pc.gamespy.com/pc/close-combat-first-to-fight/531637p1.html, July 2004.

[Tesauro, 1995] Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

[Tibshirani *et al.*, 2001] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the Number of Clusters in a Dataset via the Gap Statistic. *Journal of the Royal Statistics Society (Series B)*, pages 411–423, 2001.

[Turing, 1950] Alan. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):443–460, October 1950.

[Turn 10 Studios, 2007] Turn 10 Studios. Forza Motorsport 2. Microsoft Game Studios, http://www.forzamotorsport.net/, May 2007.

[Valve, 2000] Valve. Counter-Strike. http://www.steamgames.com/v/index.php?area=app&AppId=10&cc=CA, November 2000.

[Valve, 2004] Valve. Counter-Strike: Source. http://www.steamgames.com/v/index.php?area=app&AppId=240&cc=CA, November 2004.

[Valve, 2008] Valve. Counter-Strike: Source Screenshots. Accessed July 28 2008, http://www.steamgames.com/v/index.php?area=thumbnails&AppId=240, 2008.

[Vesterby, 2002] Tore Vesterby. Speak Softly and Carry a Big Gun: A Case Study of Professional Danish Female Counter-Strike Players. Master's thesis, The IT University of Copenhagen, 2002.

[Watkins, 1989] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

[Wray *et al.*, 2004] Robert E. Wray, John E. Laird, Andrew Nuxoll, Devvan Stokes, and Alex Kerfoot. Synthetic Adversaries for Urban Combat Training. In *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, pages 923–930, San Jose, California, 2004. AAAI Press.

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

[Yang *et al.*, 2001] Jing Yang, Simon Liao, and Mirek Pawlak. On a Decomposition Method for Finding Winning Strategy in Hex Game. In *The International Conference on Application and Development of Computer Games in the 21st Century*, pages 96–111, 2001.

[Zanetti and Rhalibi, 2004] Stephano Zanetti and Abdennour El Rhalibi. Machine Learning Techniques for FPS in Q3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE-04)*, pages 239–244, New York, NY, USA, 2004. ACM.

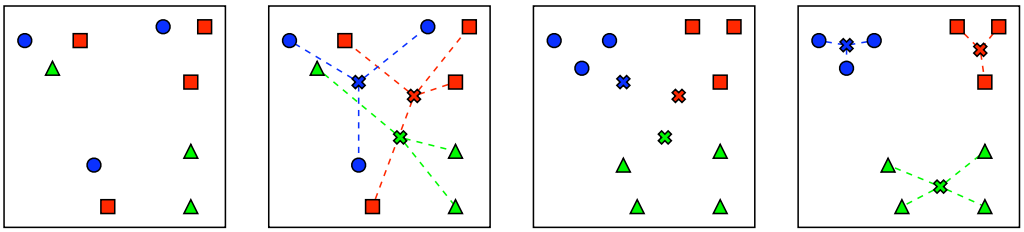
Appendix A

Methods

This appendix provides a brief overview of some of the methods employed in this study.

A.1 K-means Clustering

The K-means clustering algorithm (Algorithm A.1.1), due to MacQueen [1967], partitions points into spatially related subsets called **clusters**. A cluster is defined by its **centroid** (Definition A.1.1); a point belongs to the cluster whose centroid is nearest. K-means optimises the centroids's positions to minimise the mean squared distance between a centroid and its assigned points. This is achieved with an iterative process until convergence.[1] Hastie *et al.* provide a good introduction to the K-means algorithm [2001, p.461].



(a) (b) (c) (d)

Figure A.1: K-means iteratively refines K clusters (here, a point's shape indicates its membership to one of $(K = 3)$ clusters) which are defined by a centroid point (illustrated here with an $\times$). Each data point is initially assigned randomly to a cluster (a). Next the centroid of each cluster is calculated (b), and each point is moved to the cluster whose centroid is nearest (c). The process repeats until converging on a locally optimal solution (d).

Definition A.1.1 (Centroid). The centroid of a set of points G in n-dimensional space is the point $c \in \mathbb{R}^n$ that minimises the mean squared distance between c and each $g \in G$.[2]

[1] Convergence does not guarantee optimality, as the K-means algorithm can get caught in local optima. The implementation of k-means and k-medians tested convergence over 20 random restarts.

[2] A familiar example of a centroid is G's arithmetic mean in Euclidean metric space.

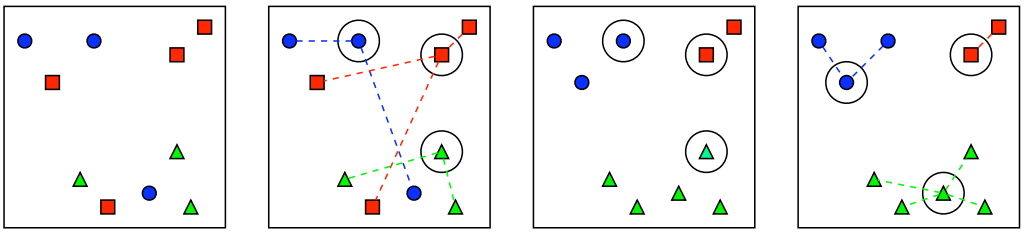
Algorithm A.1.1 K-means clustering.

Require: D is a set of point objects in $\mathbb{R}^n$.
Require: K is a positive integer.
1: Randomly divide D into K non-intersecting subsets $G_1, G_2, \ldots, G_k$ (Figure A.1(a)).
2: **repeat**
3: **for** i from 1 to K **do**
4: Define c_i to be the centroid of the corresponding cluster G_i (Figure A.1(b)).
5: **end for**
6: **for all** $d \in D$ **do**
7: Assign d to the cluster G_i with the nearest centroid c_i (Figure A.1(c)).
8: **end for**
9: **until** each of the K centroids converge (Figure A.1(d)).

K-means is quick and efficient in practice [Michie *et al.*, 1994], but Arthur and Vassilvitskii [2006] show that pathological worst-case point arrangements can result in super-polynomial run-time. Additionally, as Bermejo and Cabestany [2001] point out, outliers can have a disproportionate effect on centroids due to the squaring of distance values.

A.2 K-medoids Clustering

An alternative to K-means is K-medoids [Kaufman and Rousseeuw, 1990]. As with K-means, K-medoids is efficient in practice, but can additionally be less sensitive to outliers. This reduced sensitivity is due to the difference between minimising the mean error between a set of points and their prototype (as is done with K-medoids) and minimising the the mean *squared* error between a set of points and their prototype (as is done with K-means).



(a) (b) (c) (d)

Figure A.2: K-medoids iteratively refines K clusters which are defined by a medoid. Each data point is initially assigned randomly to a cluster (a). Next the medoid of each cluster is calculated (b), and each point is moved to the cluster whose medoid is nearest (c). The process is repeated (d) until converging on a locally optimal solution.

The K-medoids algorithm (Algorithm A.2.1) resembles K-means except that, instead of specifying a centroids as cluster centres, it specifies **medoids** (Definition A.2.1). Hastie *et al.* provide a good introduction to the K-medoids algorithm [2001, p.468].

Definition A.2.1 (**Medoid**). The medoid of a set of points G in n-dimensional space is the point $m \in G$ whose mean distance[3] from the other points $g \in G$ is minimised:

$$\underset{m \in G}{\operatorname{argmin}} \left(\sum_{g \in G} \operatorname{dist}(m, g) / |G| \right). \tag{A.1}$$

Algorithm A.2.1 The K-medoids Clustering Algorithm

Require: D is a set of point objects in $\mathbb{R}^n$.
Require: K is a positive integer.
1: Randomly divide D into K non-intersecting subsets $G_1, G_2, \ldots, G_k$ (Figure A.2(a)).
2: **repeat**
3: **for** i from 1 to K **do**
4: Define m_i to be the medoid $\underset{m_i \in G_i}{\operatorname{argmin}} \left(\sum_{g \in G_i} \operatorname{dist}(m_i, g) \right)$ (Figure A.2(b)).
5: **end for**
6: **for all** $d \in D$ **do**
7: Assign d to the cluster G_i with the nearest medoid m_i (Figure A.2(c)).
8: **end for**
9: **until** each of the K medoids converge (Figure A.2(d)).

Compared to K-means, the K-medoids algorithm requires more computation time due to the additional effort of locating the medoid in the data [Han and Kamber, 2006].

A.3 Cross-Validation

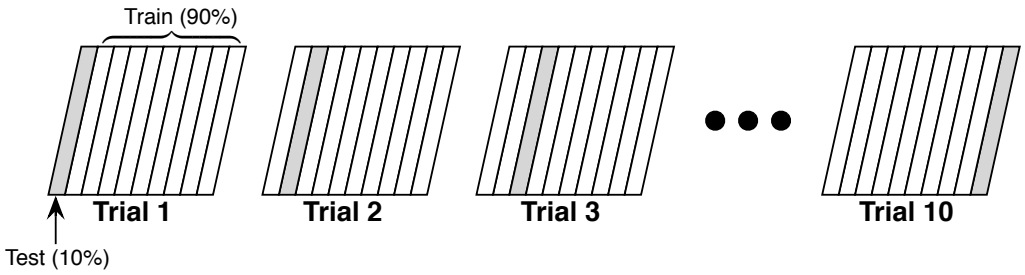


Figure A.3: Visualisation of the cross-validation process.

In ten-fold cross-validation [Kohavi, 1995], a model is repeatedly trained on 90% of the data and tested on the remaining 10%. The process begins anew for each of ten trials so that the model is tested on every element in the dataset at least once (Figure A.3).

[3] Since $|G|$ is a constant, it can be removed from Equation A.1, as applied in Algorithm A.2.1.

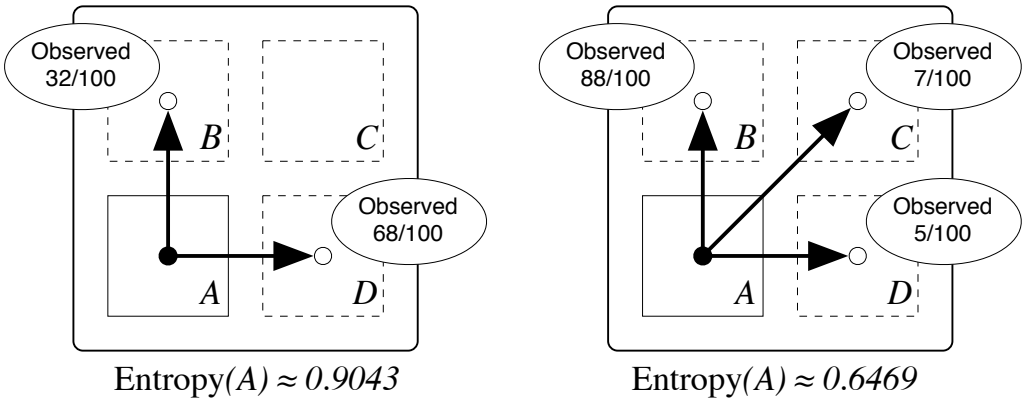


Figure A.4: Example measurements of entropy for two different sets of observations. It is worth noting that, even though there are fewer *unique* successors in the figure on the left—2 as compared to 3—the entropy level is higher because the transitions are less predictable.

A.4 Measuring Entropy in Trajectory Data

Intuitively, information entropy is a measurement of unpredictability. Entropy is used in this study to visualise some of the differences between human- and computer-generated trajectories in specific discrete locations. The less predictable players' forthcoming actions appear, the higher the entropy value associated with those players' locations.

Measuring entropy in trajectory data requires a uniform discretisation of an environment into a uniform grid.[4] Each grid cell keeps a record of successor grid cells (i.e., cells that players were observed travelling to) given as $s_1, s_2, \ldots, s_n$, and sampled transition probabilities derived from how often a successor was observed being travelled to relative to other successors, $P(s_1), P(s_2), \ldots, P(s_n)$. With these variables, the entropy associated with a particular grid cell is given as:

$$\sum_{i=1}^{n} P(s_i) \log_2 \left(\frac{1}{P(s_i)} \right). \tag{A.2}$$

Example entropy values are shown in Figure A.4, which illustrates differences in computed values for two different sets of observations. Heatmaps depicting entropy values in Valve's *Counter-Strike:Source* can be found in Figure 4.4 on page 31.

[4]Environments can also be discretised in three or more dimensions and into non-uniform segments, depending on how the trajectories are realised in the environment.

Appendix B

The Data

The *Counter-Strike: Source* [Valve, 2004] (*CSS*) human player data in this study was collected from a large competitive tournament called Fragapalooza [2008] by the Alberta Ingenuity Centre for Machine Learning (AICML) on July 2006 and July 2007. This appendix provides additional details on the quality control and formatting of these logs.

B.1 Additional Details and Quality Control

The number of unique positions in the logfiles provides a very conservative approximation of the number openings that can possibly be enacted in *CSS*; since players visit 28974 unique positions during the opening phases of play, and there are 5 players per team, there may be approximately $28974^5 \approx 2^{22}$ unique ways for such a team to open a game.

Some game logs were removed from the dataset, and are characterised as follows:

- One or both teams do nothing for the duration of a round.
- Fewer than five players per team are connected for the match's duration.
- Someone damages his/her teammates before 10 point measurements are taken.
- The game stops before an outcome can be determined.

Omitting these logs provides no guarantees of quality, but provides some assurances and succeeds in removing obvious outliers from the dataset.

B.2 Log Format

Listing 1 displays the definition of the logfile format in which the logfiles were recorded by the AICML [Grajkowski, 2006]. Note that spatial trajectory data represents only a fraction of the type of data stored in these logfiles. Listing 2 displays a sample logfile.

Listing 1 Logfile Specification

```
(this specifies the frame number for following position updates)
#<#frame number>

(notices that will probably only appear once)
!<notice>[,<args>[,...]]
  ROUND_START,<mapname>
  FREQUENCY,<#update frequency>
  TICKRATE,<#tickrate>
  ROUND_END,<#winning team>,<# win type>
    win types:
    1: bomb explodes
    7: bomb defused
    8: counter-terrorists shot all the terrorists
    9: terrorists shot all the counter-terrorists
    10: round draw

*<event>[,<args>[,...]]
  KILL,<#frame>,<#victim id>,<#attacker id>,<#weapon>, \
    <#victim pos.x>,<#victim pos.y>,<#victim pos.z>, \
    <#attacker pos.x>,<#attacker pos.y>,<#attacker pos.z>
  HURT,<#frame>,<#victim id>,<#attacker id>,<#weapon>,\
    <#victim pos.x>,<#victim pos.y>,<#victim pos.z>, \
    <#attacker pos.x>,<#attacker pos.y>,<#attacker pos.z>
  QUIT,<#frame>,<#player id>
  ERROR
  SHOT
  ZOOM
  BOMB_PLANT,<#frame>,<#site>,<#something?>,<#pos.x>,<#pos.y>
  BOMB_PICKUP,<#frame>,<#player id>
  BOMB_DROPPED,<#frame>,<#player id>,<#pos.x>,<#pos.y>,<#pos.z>
  THROW,<#frame>,<#player id>,<#weapon>,<#pos.x>,<#pos.y>,<#pos.z>
  DETONATE,<#frame>,<#player id>,<#weapon>,<#pos.x>,<#pos.y>, \
    <#pos.z>
  ROUND_NOTE,<#first frame>,<#display length>,<#pos.x>,<#pos.y>, \
    <String note>
  HOSTAGE_FOLLOWS,<#frame>,<#player id>,<#hostage id>
  HOSTAGE_STOPS_FOLLOWING,<#frame>,<#player id>,<#hostage id>
  HOSTAGE_HURT,<#frame>,<#player id>,<#hostage id>
  HOSTAGE_KILLED,<#frame>,<#player id>,<#hostage id>
  HOSTAGE_RESCUED,<#frame>,<#player id>,<#hostage id>,<#site id>
  HOSTAGE_CALL_FOR_HELP,<#frame>,<#hostage id>
  HOSTAGE_RESCUED_ALL,<#frame>

(player position updated)
<#player id>,<name>,<network id>,<#team>,<#weapon>,<#health>,
    <#pos.x>,<#pos.y>,<#pos.z>,<#angle.x>,<#angle.y>,<#angle.z>

(hostage position update)
<#hostage id>,,<#pos.x>,<#pos.y>,<#pos.z>
```

Listing 2 A Sample CSV File

```
!VERSION,120
!ROUND_START,de_dust2
!FREQUENCY,45
!TICKRATE,100.000002
#45
18,[name],STEAM_0:1:9531171,2,12,100,-648,-824,135,0,46,0
34,[name],STEAM_0:1:10535659,3,28,100,160,2304,-126,0,-104,0
24,[name],STEAM_0:0:5114862,2,5,100,-552,-920,149,0,53,0
25,[name],STEAM_0:0:11810900,3,28,100,160,2144,-126,0,-180,0
22,[name],STEAM_0:0:3296685,2,12,100,-552,-824,133,0,11,0
36,[name],STEAM_0:1:2599255,2,12,100,-552,-736,135,0,1,0
26,[name],STEAM_0:1:200791,3,28,100,256,2144,-126,0,-103,0
28,[name],STEAM_0:0:7643029,3,28,100,352,2144,-126,0,-15,0
29,[name],STEAM_0:1:4118538,2,12,100,-648,-736,138,0,0,0
33,[name],STEAM_0:0:4548289,3,28,100,448,2144,-126,0,-15,0
*BOMB_PICKUP,46,22

...

#6525
25,[name],STEAM_0:0:11810900,3,28,48,-1350,2266,3,0,115,0
22,[name],STEAM_0:0:3296685,2,12,100,-1611,2652,3,0,-56,0
29,[name],STEAM_0:1:4118538,2,5,26,-1975,1659,32,0,35,0
*HURT,6543,25,22,12,-1355,2269,3,-1611,2652,3
#6570
25,[name],STEAM_0:0:11810900,3,28,29,-1430,2305,4,0,119,0
22,[name],STEAM_0:0:3296685,2,12,100,-1611,2652,3,0,-56,0
29,[name],STEAM_0:1:4118538,2,5,26,-1977,1663,32,0,36,0
*HURT,6577,22,25,28,-1611,2652,3,-1437,2313,5
*KILL,6577,22,25,28,-1611,2652,67,-1437,2313,5
#6615
25,[name],STEAM_0:0:11810900,3,28,29,-1475,2368,5,0,115,0
29,[name],STEAM_0:1:4118538,2,5,26,-1992,1749,32,0,36,0
#6660
25,[name],STEAM_0:0:11810900,3,14,29,-1514,2450,11,0,111,-0
29,[name],STEAM_0:1:4118538,2,5,26,-1959,1838,1,0,53,0
#6705
25,[name],STEAM_0:0:11810900,3,14,29,-1550,2551,7,0,85,0
29,[name],STEAM_0:1:4118538,2,5,26,-1943,1855,1,0,57,0
#6750
25,[name],STEAM_0:0:11810900,3,14,29,-1553,2654,2,0,37,0
29,[name],STEAM_0:1:4118538,2,5,26,-1943,1855,1,0,57,0
#6795
25,[name],STEAM_0:0:11810900,3,14,29,-1464,2731,3,0,16,0
29,[name],STEAM_0:1:4118538,2,5,26,-1929,1858,1,0,61,0
#6840
25,[name],STEAM_0:0:11810900,3,14,29,-1367,2745,5,0,-123,0
29,[name],STEAM_0:1:4118538,2,5,26,-1875,1897,5,0,61,0
!ROUND_END,2,1, #Target_Bombed
```

Appendix C

Resources

This appendix contains information on select resources related to my thesis. Specifically, it describes the hardware and software I used to develop my results, pointers to the online services used to perform statistical analysis, and web addresses to directly related content.

C.1 Experimental Environment

My experimental platform consists of two machines that were provided by the IRCL research group. The first machine, 'compeer', is an AMD Athlon 64 X2 Dual Core Processor 4400+, and was used to develop the results in both Figure 4.4 and Section 6.2. The second machine, 'crestomere', is an AMD Athlon MP 1800+ (dual core) and was used to develop the results in Sections 6.3 and 6.4. Both machines ran Java version "1.5.0_06".

C.2 Statistical Utilities

The statistical analysis in this dissertation was assisted by some publically available online resources. In particular, I would like to acknowledge the use of Graphpad Software, Inc.'s free statistical calculators [2005], which were used to develop the categorical data analysis and the t-test comparison of mean values that were presented in Section 6.2.

C.3 Related Links

The following web pages may indicate future developments or errata related to this thesis:

1. The author's web page: http://www.cs.ualberta.ca/~rayner/
2. The author's research group web page: http://ircl.cs.ualberta.ca/
3. The CSAI research group web page: http://ircl.cs.ualberta.ca/games/cs/