Cooperative Strategies in Competitive Multi-Agent Environments

Competitive Path-Finder Robots

Vince Trencsenyi[1]

MSc Robotics and Computation

Prof. Stephen Hailes

Submission date: 31 August 2019

Abstract

We propose an uncertain multi-agent environment where node interaction can be investigated under both cooperative and competitive circumstances. We approach inter-agent interaction via Game Theory. We assign Robots - represented by intelligent agents in the proposed simulation framework - to the task of reaching an undisclosed location in the environment, which we represent by a maze. The destination can be given as a shared goal for all agents or separate goals can be assigned. The task is approached as cooperation problem where the participants attempt to maximise the collective payoff, then the results are compared with a competitive approach where agents are purely self-interested.

We introduce a multi-agent framework capable of hosting umpired iterated games to simulate the proposed environment, which is used to conduct experiments and evaluate the emerging behaviours as a result of different cooperation strategies and learning agent methods. This system is used to examine and reason about these interaction strategies and the resulting emergence of norms.

Contents

Chapter 1

Introduction

-intro to the topic
-problem formulation: path finding problem + conflict situations(info sharing, meeting other players) -essay structure

Contributions

- independent work

This is just a bare minimum to get started. There is unlimited guidance on using latex, e.g. `https://en.wikibooks.org/wiki/LaTeX`. You are still responsible to check the detailed requirements of a project, including formatting instructions, see `https://moodle.ucl.ac.uk/pluginfile.php/3591429/mod_resource/content/7/UGProjects2017.pdf`. Leave at least a line of white space when you want to start a new paragraph.

Mathematical expressions are placed inline between dollar signs, e.g. $\sqrt{2}, \sum_{i=0}^{n} f(i)$, or in display mode

$$e^{i\pi} = -1$$

and another way, this time with labels,

$$A = B \wedge B = C \rightarrow A = C \tag{1.1}$$

$$\rightarrow C = A \tag{1.2}$$

note that

$$n! = \prod_{1 \leq i \leq n} i \tag{1.3}$$

$$\int_{x=1}^{y} \frac{1}{x} \mathrm{d}x = \log y \tag{1.4}$$

We can refer to labels like this (1.1).

Chapter 2

Background

2.1 Agents

It is difficult to define *agents* in the context of computer science, partly due to the term being overloaded and partly because there has not been an official and agreed upon definition to encapsulate everything that is an Agent. Following the introduction in [Nwa96], we can trace back the origin to 1977, when Carl Hewitt proposed a specific object model capable of concurrent interaction. He classified such objects as *actors*. By his definition, actors can communicate via messages and they execute concurrently. By a broader definition, we can define agents as computer programs performing user delegated tasks, without user intervention. In the context of our work, the notion of *Intelligent Agents* is of more interest.

2.1.1 Intelligent Agents

Stuart J. Russel and Peter Norvig defined intelligent agents to take the best possible action in any situation. In their book [Rus03] they define agents as anything that can perceive its environment through its *sensors* and manipulate its environment using its *actuators*. Agents in this sense can be robots, devices, software, but even humans fit the description. Expanding on the terminology, a *percept* is any input the agent receives at a given time through its sensors and the *percept sequence* denotes the history of everything the agent has perceived. Between the sensors and the actuators there is a decision making process to determine the agent's choice of action, which we call the *agent function*. The agent function is an abstract mathematical concept which maps every percept sequence to an action. Unless bounded - or in the case of simplistic agents -, the number of mappings is infinite. The *agent program* implements and executes the agent function in practice for any artificial agent.

[Rus03] A *rational agent*'s agent function table entries are all correct, meaning the agent will execute the right action for any given percept. However, we need a definition of what is the "right thing" to do. Agents act upon the perceived environment, changing its state. We would consider the agent to have performed well, if the sequence of state changes in the environment is desirable. The perfomance measure we can use to evaluate the behaviour of an agent is based on this

desirability. However, the perfomance measure must be designed to fit the specific application, accommodating the circumstances. We can then define rational agents as agents, whose choice of action - based on the evidence acquired from the percept - is expected to maximize the perfomance measure for each possible percept sequence.

Characterized by their agent program, [Rus03] classifies agents into four categories:

Reflex Agents are the simplest examples, they base their decisions solely on the current percept. Reflex agents function following condition-action rules, which are relation in the form of "if - then " statements. For example a thermostat agent would have a behaviour like: *"**If** temperature is below 20C, **then** start the heating ".*

In order to be able to handle more complex problems, the agent might need to have at least some representation of unobserved information in addition to its current percept - unless the environment is fully observable. It needs to maintain an internal state based on the percept history. It can then develop knowledge on how the environment is changing state - without the agent's own influence - and how the action of the agent affect the environment. These two details make up the agent's model of the "world". A *Model-Based Agent* uses such model to aid its decision making.

Goal-based Agents, in addition to a description of the current environment state, have a description of the desirable state of the environment to be reached. These states are the goals of the agent. Goals provide an extra layer of flexibility. While changes in the environment would require the designer to change the condition-action rules of the Reflex agent, a goal-based agent can adopt these changes updating its knowledge and can derive the optimal choice of action.

In most environments, achieving the goal alone does not guarantee maximized performance measure. There might be multiple ways or methods for an agent to reach its goal, out of which some are more desirable than others. An agent that uses utility to describe the measure of how favorable an outcome is and uses an utility function to maximize the utility it gains is an *Utility-based agent.*

2.1.2 Task Environment

We have introduced the notion of agents by describing its sensors, actuators, the environment it acts upon and the performance measure to evaluate its behaviour. [Rus03] These characteristics together make up the *task environment*, which is used to define and discuss agents. Alternatively, the term *PEAS description* is also used - an acronym of the starting letters of Performance Measure, Environment,Sensors, Actuator. The task environment can be categorized by a number of characteristics that determine the appropriate agent design and methodology.

Observability characterizes how much information the agent can acquire about the environment. If the agent has no sensors, then the environment is *Unobservable.* If the sensors can

provide complete information about the state of the environment at any given time - or about everything that is relevant to the agent's decision making process -, then the environment is *Fully Observable*. Otherwise, we call the task environment *Partially Observable*.

If the agent's choice of action and the current environment state determine the next environment state explicitly, then the task environment is *Deterministic*. If the uncertainty about the possible outcomes is measured in probabilities, then it is a *Stochastic* environment. Otherwise, *Non-deterministic*. Furthermore, we call an environment *uncertain*, if it is not fully observable or is not deterministic.

In an *episodic* task environment the sequence of interaction of an agent is divided into single episodes. The decision making in each episode depends on the current percept, previous episodes do not affect the choice of action. In a *sequential* environment on the other hand, actions could affect future choices. Therefore, the agent has to think ahead.

If the state of task environment does not change during the agent's decision making and the passage of time is not relevant, then it is a *static* environment. In dynamic environments the state is continuously being updated, regardless if the agent has acted upon it or not. In a *semi-dynamic* environment the environment state does not change with the passage of time, but the agent's performance score does.

We also distinguish environments based on how the states,percepts and actions are handled with relation to the passage of time. In a *discrete* environment there are a finite number of distinct states, so the inputs and the actions of the agent are also discrete. In a *continuous* task environment the percept of the agent represent the continuous changes in the state of the environment and its action is carried out in a continuous manner as well.

An environment is *known* if for each action the corresponding outcome is given, so the agent knows the consequences of its decision. In an *unknown* environment the agent does not know the results of certain actions, it has to learn the "rules" of the environment to be able to make efficient choices.

The difference between a *single-* and *multi-agent* environment is as straightforward as the type names suggest. However, it is not always as simple to determine when entities in the environment should be handled as agents or as objects. In general, an object whose behaviour can be described as performance maximizing must be handled as an agent. Multi-agent environments can be *co-operative* - when the actions of the agents maximize the collective performance - or *competitive* - when the action of an agent maximizes its own performance while minimizing other performances.

2.1.3 Multi-Agent Systems

As briefly discussed in the previous section, a Multi-Agent Environment contains multiple agents. Similarly, [Woo02] a Multi-Agent System comprises of multiple agents interacting with each other through communication. These agents are simultaneously making choices, affecting the environ-

ment and the other agents. We discuss Interaction and Cooperation via Game Theory later, in section 2.3.

[Woo02] Interaction between self-interested agents can be problematic, especially if we expect cooperative behaviour. An approach to handle such situations is Reaching Agreements. A fundamental aspect of intelligent autonomous agents is the ability to reach agreements without a controlling authority. In this context this can be realized through *negotiation* and *argumentation*. A *Mechanism design* contains the protocols that govern the agent interactions. These negotiation protocols should optimally be designed to have the following desirable properties:

- A protocol should ensure that an agreement is reached, thus providing *Guaranteed Success*.
- A protocol should *Maximize Social Welfare*, maximize the collective utility of the agents participating the the negotiation.
- Negotiation outcomes should be *Pareto Efficient*, where it is not possible increase an agent's utility without decreasing an other's.
- To motivate participation in negotiations, a protocol should be designed in such a way, that it is in the best interest of the agents to follow it. Such protocols are *Individually rational*.
- A *stable* protocol provides an incentive for a certain behaviour for all agents.
- A *simple* protocol allows the agent to easily derive an optimal strategy.
- A protocol maintaining the *distribution* property does not contain a "single point of failure" to minimize the communication between agents.

In fact, it is relatively simple to design protocols that provide the aforementioned properties, which allows the application of Game Theory to evaluate negotiation techniques.

Communication

Inter-Agent interaction in Multi-Agent systems is realized via communication. While with Object-Oriented Programming one object could invoke another object's public method or change its public variables as a realization of communication, this is not viable in the case of Agents. [Woo02] Agents are autonomous entities running simultaneously and continuously, their internal state can not be changed externally, nor can they be forced to execute a certain action. What agents can do, however, is to conduct communicative actions to influence the other agent.

[Woo02] The speech act theory states that communicative acts should be handled like any other action the agents may perform. Following the work of the John Austin, fellow Philosopher J. Searle identified properties required for any communication act to be successful between the speaker and the hearer. [Sea69] For example, a request would have properties such as:

- *Normal Input/Output conditions* state, that the communicative act was performed under normal circumstances and that the hearer can receive the communication without obstruction.

- *Preparatory Conditions* are conditions that must be met before the act can be carried out successfully. In the case of a request, the speaker must have the belief that the hearer can perform the requested action, the hearer indeed must be able to perform the act and it must not be obvious that the hearer would perform the requested action anyways, without the request taking place.
- *Sincerity conditions* are used to determine whether the speaker intended the performance of the act.

Searle also attempted to classify speech acts into five main categories. *Representative* acts are used to provide information, *Directives* are used to request the hearer to perform some action, the speaker commits itself to a course of action via a *Commissive* act, *Expressive* acts are used to express a psychological state of the speaker and *Declarations* are used to issue changes in the circumstances with relation to the speaker and the hearer.

The speech act theories motivated the design of several communication architectures. [Woo02] The Knowledge Query and Manipulation Language is a language designed to provide a common format for agent communication. KQML has no tools to reason about the contents of the message, it is mainly concerned with the structure of message objects. In KQML, each message is defined by its performative class and by a number of parameters. Some of the more important parameters are: *:content*, containing the "message" itself; *:sender* and *:receiver* marking the origin and intended destination of the message; *:language* denotes the language of the message content, which the recipient is assumed to understand; *:ontology* is the terminology used in the message content. The table below lists a few examples of the 41 different performatives identified in the 1993, Finin et. al version of KQML.

Performative	Meaning
achieve	S wants R to make something true of their environment
broadcast	S wants R to send a performative over all connections
discard	S will not want R's remaining responses to a query
forward	S wants R to forward a message to another agent
pipe	S wants R to route all further performatives to another agent
reply	communicate an expected response

Figure 2.1: Possible KQML performatives

While KQML is used to determine the format of the message, the Knowledge Interchange Format describes the content of the message. KIF is based on first-order logic and implements the fixed-logic apparatus of the general boolean connectives and existential and universal quantifiers - e.g. and, or, not, for all, exists. Furthemore, KIF has a vocabulary of objects, such as numbers and characters, and corresponding object relations - e.g. greater/lesser than. These features allows the definition of objects and relations, and the expression properties of "things" in the domain, the relationship between "things" in the domain and general properties in the domain.

[Woo02] KIF is also considered to be a language which can be used to define ontologies. An

ontology ensures that the communication participants use and understand the same terminology in their messages. However, KIF was designed to be processed by computers, it was not intended to be a language for the human users. The Ontolingua server is a software tool, which provides a common platform where different ontologies can be shared. Ontologies are defined in the KIF based Ontolingua language and then stored in an online library.

Game-based architecture

Stathis and Segot [SS96] proposed an approach to developing interactive system, which handles interaction and functionality as games. They use this metaphor to define a conceptual framework for developing complex interactive systems. Their work is focused on simplifying Knowledge-Based Front-Ends, however, the proposed modular concept can be used as a base for developing a multi-agent system, given that interaction between agents can also be evaluated as players playing games.

[SS96] considers the rules of an interactive system as the rules that define a game. Following the metaphor, the actions of the participants of the system are handled as moves selected by players of a game. The available moves in the game are specified by the preconditions of interactions in the system and the effects of these moves on the game state correspond to the changes in the system. In addition to the players, they utilize an extra component. The *umpire* is responsible for displaying the state of the game and organizing interaction between players. The use of umpires introduce a variation of plays, thus allowing networked games. This feature makes the concept compatible with applications where participants are physically not in the same location - Fig. 2.2 displays the similarity between the model of interaction of a game and an interactive system .

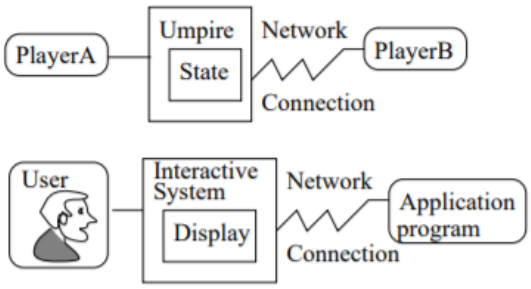

Figure 2.2: [SS96] "Playing a Game vs Interacting with an Interactive System"

The motivation for a game-based approach is the simplicity with which we can define systems as games. A complex interactive system can be represented as compound games composed of simpler sub-games, corresponding to different separable application functionalities or interactions the user can perform. An other useful property the concept offers is the implementation of advisors. Since interaction is handled as playing moves in a game, the use of strategies can be applied. An advice-giving component can formulate strategic and context sensitive advice, carried out as an optimal interaction strategy - as in Game Theory.

The metaphor introduced in [SS96] is extended in [Sta00] to support the sort of interactions taking place in Multi-Agent Systems. Atomic games are utilized, defined by the com-

ponents: states, rules and moves. A move is an Act selected by a player, which can be represented in the form of $select(Player, Act)$. The game state is defined by rules of the form $rule(State, Property, Conditions)$, where $State$ denotes the specific game state, in which a $Property$ holds, if some $Conditions$ are met.

```
rule(qprotocol(qp1),player(p1),[]),
rule(qprotocol(qp1),player(p2),[]),
rule(qprotocol(qp1),about(tel(jim,123456)),[]),
rule(qprotocol(qp1),next(p2),[]),
rule(qprotocol(qp1),last(p1),[]),
rule(qprotocol(qp1),previous(select(p1,query(tel(Jim,123456)),yes)),[]).
```

Figure 2.3: [Sta00] An example of state instance definition

In Fig. 2.3 above, we reason about the state of the query protocol $qp1$. The rules governing the state defines the instance, where the participants are player $p1$ and $p2$. Player $p1$ queried $p2$, whether the telephone number of "Jim" is 123456. The state also stores information about who made the last move, what that move was and who is playing next.

System goals, rules, negotiation and properties of the game state are defined using for example *solve(State,[])* or *solveOne(State,Goal)*. To apply changes in game state or consequences of player action, *effects(State,[])* and *effectsOne(State,Action)* is used, governed by rules of the form : *effectRule(State, Action, Conditions, Actions)*. This statement means, that if the conditions of the specified action are met in the given state, then the corresponding list of actions are carried out.

The rules and the initial game state define all possible outcomes. When the game is played a single outcome of the set is reached, via player selecting moves consecutively. In an umpired game, agent interaction and behaviour is controlled by a separate component. The umpire maintains the state of the game and computes the results of the validated moves.[Sta00] If the players choose their moves from a pre-defined set, then the validity is defines in the *forward mode*. Move validation can be also executed in *backward mode* where the players do not have an explicit list of available moves. Instead, the umpire validates the move selected by a player by applying the rules that govern the game.

The focus of [Sta00] is on logical programs, which is out of the scope of this paper - nevertheless, the concept itself is applicable to our purposes. The interpretation and definition of the game components in the logical program context - rules, states, actions, the umpire, strategies, etc. - are discussed in more detail and demonstrated with examples in the original paper.

Such umpired game-based architecture is used as base for the multi-agent system we utilize to provide an environment for our maze solving nodes, further discussed in Chapter 3.

2.2 Path planning

2.3 Node cooperation

Multi-agent systems involve interaction between the individual agents. As discussed in section 2.1, agents are trying to maximize their measure of performance. As a result the interaction can become cooperative or competitive, and agents will require a strategy to handle these conflict situations during their decision making. Game Theory provides the tools to discuss and investigate such approaches and to analyze conflict.

2.3.1 Game Theory

Game Theory has been a remarkably popular field in Economics ever since Von Neumann and Morgenstern published their work in 1944, [Von44]. Not only did they introduce a revolutionary approach to economic and social organization, but they created the fundamentals for a new and widely applicable field of research, by investigating social and other forms of interaction as games of strategy.

[Mye91] defines games as any social interaction between at least two players. Game Theorists usually take the general assumption that parties participating in any interaction are intelligent and rational, which in this context means players conduct their decision-making process to maximize their pay-off towards their objective, measured by *utility*. The base principle for rationality in this context is given by Von Neumann and Morgenstern. They showed that for any decision-maker in any situation it is possible to assign a utility value to each possible outcome - that the player cares about - and that the decisions made ultimately lead to maximizing the accumulated utility. This phenomenon is called the "expected utility maximization theorem", and its roots go back to evolutionary biology: [May82] any living being behaves in a way that best supports its survival, in other words they try to maximize some measure of evolutionary success.

Game Theory provides a strong mathematical definition for games and their components: players, actions, information and utility (Payoff). As a result we have a way to reason about social interaction scientifically, allowing us to study and evaluate strategic approaches. In order to represent a game "formally", we have to define all four elements stated above, referred to as *PAPI* in [Ras07]. As we introduced before, *players* of the game are the decision-makers, *actions* are the available interactions in any given situation, *information* is the data available to the decision-makers at the time of interaction and *utility* is the unit of measure of some payoff the players are trying to maximize towards their corresponding goal or objective.

In order to reason about games we have to represent them numerically, for which we have multiple possible approaches, depending on the characteristics of our problem and requirements of our application.

Extensive Form

The extensive form represents games as decision-trees. [Ras07] Expanding on the terminology of the tree structure, a node in the tree represents a point in the game at which a decision is made

and an action is taken. Nodes have predecessors, which are nodes that must be reached before we can get to the specific node. Successors to a node A are the nodes that can possibly be reached from X. A starting node has no predecessors, and an end node has no successors. Branches are single actions for a player at a particular node, and paths represent the sequences of actions taken, leading from a starting node to an end point. Extensive form trees contain no closed loops between starting nodes and their corresponding end nodes.

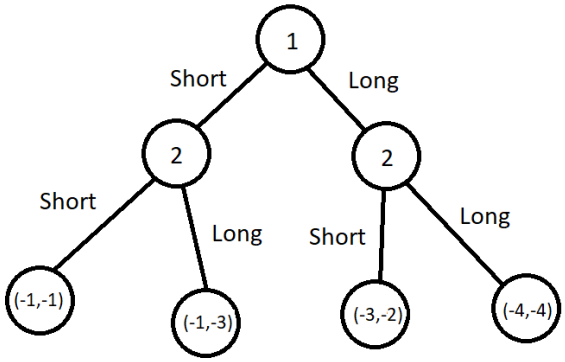

Figure 2.4: Extensive form tree, showing a one round game between two players

As Figure 2.4 shows, the nodes are labelled as the corresponding players, except for the end nodes. The edges are labelled with the corresponding action or decision and the end nodes with the resulting pay-off for the players. Consider , in a game of path planning, Player 1 chooses to take a shorter path. Player 2 - on the left branch - then selects the longer path, which results in the pay-off of (-1,-3) - (Player 1, Player 2). As the goal is to maximize the pay-off both the individual and collective optimal outcome is if both players took the shorter path.

[Ras07] describes an extension to the graph introduced above, which allows us to represent the information states of the player, i.e. what information is available to a player P at the point of a specific decision-making taking place. This is done via adding dotted lines representing the set of available information. [Mye91] also shows that labels at decision nodes can be used to provide more information on the states. A node with label *1.b* depicts that Player 1 is in information state b at the time of taking action. [Mye91] In this context, a *strategy* is a function mapping information states into decisions. $X_{s \in S_i} D_{S_i}$ denotes the set of available strategies for player i, where S_i is the set of information states and D_s is the set of moves for player i in information state s of S_i.

[Ras07]An alternative notion to the Extensive Form is the Game Tree. Game Trees are identical to the Extensive Form, except that the end nodes contain the *outcome* instead of *the pay-off*.
The Time line is an alternative representation of games, more frequently used for Economics applications. A time line shows the sequence of actions in their order of happening, but not necessarily the passage of time.

Strategic Form

[Mye91] describes the Strategic Form as a simpler method of representing games:

$$\Gamma = (N, (C_i)_{\in N}, (u_i)_{\in N}) \tag{2.1}$$

where N is the set of players in the game, C_i is the set of strategies available for player i and u_i is a function mapping from $X_{j \in N}$ to R. Given $C = X_{j \in N} C_j$, $u(c)$ is the expected utility for player i implementing the set of strategies c. In the strategic form we usually assume simultaneous player decision, therefore the element of time can be ignored when the games are analyzed. While the extensive form provides a dynamic model with total information about how the sequence of action take place over time throughout the game, the strategic form provides a simpler representation where timing is not an important factor.

Strategic form games can be represented using the *normal representation*, shown below.

-	Player 1: A	Player 1: B
Player 2: A	1:1	0:2
Player 2: B	2:0	1:1

Figure 2.5: Normal representation of a game, Payoffs (P1 : P2)

This allows us to represent games in matrix form, often called a Payoff Matrix. The inner cells of the matrix show the expected payoff resulting from the combination of choices of Player 1 and Player 2. If Player 1 utilizes its strategy A and Player 2 uses B, the resulting Payoff is 2 for A and 0 for B.

[Von44] states that in a general sense, the normal form is all we need to evaluate. The argument is based on the general assumption we stated earlier, that all players are intelligent and rational in their decision making. If this is the case, both the player and the theorist can compute the rational and optimal paths of actions ahead of the game. Thus, we do not lose generality with the assumption that the players formulate their actions simultaneously before the game begins. The sequence of the game then just becomes the mechanical application of player strategies and the determination of the final payoff according to the game specific rules. In this context the normal form gives us all the required information.

Characteristic Function form

The characteristic function is defined in [Bar49] as the largest possible payoff to a coalition S. Let us denote a coalition as a subset $S \subset N$, where N is the set of all players. Then any function $v : 2^N \to R$ satisfying $v(\emptyset) = 0$ and $v(N) = \sum_{i=1}^{n}$ is said to be a characteristic function of an n-person cooperative game.

The characteristic function form is mostly used to define cooperative games - as the definition suggests - as it allows us to take into account player coalitions. It is also useful for scenarios where utility is removable and where individual rewards are not assigned, in which case we can determine the payoff for each outcome with characteristic functions.

Game Types and Characteristics

[Bar49] Cooperative games allow players to form coalitions, where they may base their strategies on cooperating with other players, thus maximizing their common payoff. Non-cooperative games provide a more general approach to investigate individual decision making strategies and they focus on achieving Nash-equilibrium.

[May82] introduces the difference between symmetric and asymmetric contests. In symmetric games players have common strategies and the corresponding utility is not dependent on the identity of the decision maker, a certain action yields the same reward for any player. Asymmetric games on the other hand assign different utility values for the same action, but of different players - or the corresponding strategies may differ. Figure 2.5 represents a symmetric game, while Figure 2.7 shows an asymmetric payoff assignment.

-	Player 1: A	Player 1: B
Player 2: A	1:3	1:2
Player 2: B	2:0	2:1

Figure 2.6: Asymmetric payoff matrix, Payoffs (P1 : P2)

Zero-sum games, as defined in [Von44], have a special condition on their payoff matrices, that the sum of utility values for each pairing of player decisions is zero. Thus, in a Zero-sum game a player can only gain benefit at the cost of the other contestant. The payoff matrices we have used so far were all of general, non-zero sum games. The figure below shows the utility payoffs of a Zero-sum game.

-	Player 1: A	Player 1: B
Player 2: A	0:0	2:-2
Player 2: B	-2:2	0:0

Figure 2.7: Zero-sum game, Payoffs (P1 : P2)

We can classify games based on the amount information available for players in the game. [Ras07] describes information sets as the set of different nodes in the game tree for a player, such that the player knows the actual node is in the set but can not determine exactly which by direct observation. An information set contains nodes belonging to a single player, but on multiple paths. In other words, the player knows whose turn it is, but does not know the exact node reached in the game tree. An information partition is a collection of a player's information sets, where each path is represented by a singleton and predecessors of nodes in the same information set are also in a common information set.

Perfect information games contain only singleton information sets, otherwise the game is of *imperfect information*. Essentially, in a perfect information game all players know the previous decisions made that affect them and led to the current node.

In *symmetric information* games, any player p at an end node or decision node has at least as many elements in its information sets as there are in the other players'. In any other case, the game is with *Asymmetric information*. The notion of Asymmetric games allow us to define *private information*, which represents *"an information partition that is different and not worse than another players"*.

In an *uncertain* game, some moves are not immediately revealed to the players - e.g. by Nature -,while in a game of *certain*ty Nature does not have any moves after the players move. *Nature*, as defined by Rasmusen, is a pseudo-player making random decisions at specific points in the game.

A game has *complete* information, if knowledge about the players - their strategy, utility function, payoff - is available to all contenders. However, the decisions made by players does not need to be revealed. In *incomplete* information games such information is not available or not for everyone.

2.3.2 Prisoner's Dilemma

Game Theory is widely used in various fields of applications - economics, politics, biology, sociology etc. - as it provides a scientific and mathematical foundation to evaluate interactions and conduct Conflict Analysis. The Prisoner's Dilemma is a popular example of such a setting. The idea comes from an abstract example, where two criminals are being questioned in seperate rooms. They have no means of communicating with each other and they do not know what the other suspect will do. Their choices are based on the assumption that they will never meet again and there is no retaliation to expect. Their individual sentences depend on the combination of their choices. There are three possible outcomes: if both suspects cooperate they both get a lower sentence; if both of them defect they get a more severe sentence; if one cooperates and the other defects, the defector can walk free while the suspect that cooperated gets the most severe sentence of the situation.

[May82] Cooperation proposed a problem even for Darwinists. Darwin recognised that kinship has a key role in emerging cooperation - within or even between species. Through a slow progress of research on cooperation, we now know that both kinship and mutual benefits are causes for social behaviour to evolve into cooperation among individuals.

[Axe84] Axelrod based his work in investigating cooperation. His approach is based on the social trait of selfishness in humans, originating from evolution and the "survival of the fittest". Within this setting, he was looking to find *"Under what conditions will cooperation emerge in a world of egoists without central authority?"*. Axelrod investigates how cooperation can evolve in purely self-interested individuals. He presents a Cooperation Theory in which "social welfare" is ignored, so the emergence of cooperation is not affected by any controlling authority or concern for others.

Purely selfish behaviour, however, is likely be the less rewarding option in many cases when we investigate the outcome in relation of all participants. There are situations where the individual

is better off not cooperating, but looking at both parties we find that the group of participants would have obtained a better outcome if both of them chose to cooperate. [May82] The Prisoner's Dilemma is a symmetric two-person game with only two available action for both players : Cooperate or Defect. The choices are made simultaneously, the contenders do not know the other's choice. In [Axe84] the payoff matrix for the games is described to represent the issue with self-interest over the common benefit - see Figure 2.8 below.

-	Player 1: C	Player 1: D
Player 2: C	3:3	0:5
Player 2: D	5:0	1:1

Figure 2.8: The Prisoner's Dilemma, Payoffs (P1 : P2)

If both players cooperate they are rewarded *for mutual cooperation (R)*. If one player cooperates but the other defects, the cooperating player gets the *sucker's payoff (S)* while the other receives the *temptation to defect(T)*. Thus, inspecting Player 1's choices it is better to Defect regardless what Player 2 decides to do. If player 1 defected, it would either get 5 or 1 utility versus receiving 3 or 0 points if he cooperated. Self-interest motivated social behaviour will lead to both players defecting, when it would have been mutually beneficial if both cooperated. While the magnitudes of the payoff values may change - as described later in this section - the relation between the categories must remain the same: $T > R > P > S$ and $T + S < 2R$.

The above example describes a single game of the dilemma. However, Axelrod proposed a tournament of such games, where multiple players play the games repeatedly. [Axe84] The goal of the tournament is to find the strategy that will yield the highest possible payoff in the end. In the Iterated Prisoner's Dilemma actions may lead to consequences, as the players meet again. If Player A cooperates on the first move, but Player B Defects, then A might choose to retaliate B's selfish action in the next round of the game. There are several ways to realize the Prisoner's Dilemma, but in Axelrod's tournaments - which is also the version of the game our work focuses on - the following rules apply:

- Players may interact with many other, but only one at a time
- Players can not enforce threats or commitments, thus all players have access to all strategies.
- Players can not know for sure what action the other will take at a certain move. The only information available to players about the others is their common history. There is no information about interactions with a third party.
- Decision making is inevitable, each play must choose from the available actions: Cooperate or Defect.
- It is not possible to change an other player's payoff.

[Nas51] The Nash Equilibrium is a solution concept in Game Theory for non-cooperative games. The Nash Equilibrium defines a state in the game where maintaining a strategy is in the best interest of the player, assuming the other player's strategy remains the same. In other words, a

player would not benefit from changing its strategy if the other player(s) also retained their own. This definition allows us to consider another key detail in the Iterated Prisoner's Dilemma. If the number of rounds is known by the players, then the only emerging strategy would be to Always Defect. If Player A knows it will face Player B exactly 10 times then its Nash Equilibrium becomes the Always Defect strategy. The rational decision in the 10th round is to Defect, as the opponent will not be able to retaliate. Player A will then expect B to also Defect and A's incentive to Cooperate in the 9th round diminishes because B would Defect in the next turn anyways. This line of reasoning can be backtracked for both players all the way to the initial round, at which point Always Defect becomes the Dominant Strategy for both participants.

To be able to truly examine emerging cooperation Axelrod's tournament did not reveal the number of rounds to the players, which also characterizes the environment of our research - the length of the shortest path and the number of encounters with other nodes is unknown to the contestants. [Axe84] The Prisoner's Dilemma is abstract, but it allows us to weaken some assumptions in general Game Theory, which makes it a versatile and widely applicable concept:

- It is not necessary to use a comparable unit of measure for the payoffs, allowing us to analyze players from different "domains", with different types of reward.
- While it is convenient to use symmetric payoff matrices, it is not a necessity for the concept. As the unit of measure can be different, the payoff categories can yield different magnitudes, if they maintain the defined ratios.
- It is sufficient to measure player payoffs relative to each other, absolute measurement is not necessary.
- It is not necessary to promote cooperation, strategies can be used in reverse to prevent the emergence of cooperation.
- Players do not need to be rational, their decision making can be based on instincts, habits, randomness.
- We do not need to assume deliberate choices. Some behaviour can be conducted without considering strategies.

2.3.3 Strategies

Players of the Prisoner's Dilemma use strategies for their decision making to achieve their objective - whether be that maximizing utility or some other goal. [Mat17] summarizes 17 basic deterministic and 13 probabilistic strategies. In addition to the *Always Defect* and *Always Cooperate* strategies, we can utilize a variety of *periodical* strategies executing and repeating a predefined sequence of actions - e.g. **C-C-D** or **D-D-C**.

Reactive strategies, such as *Pavlov* and *Tit-for-Tat* consider previous moves and outcomes for decision making. [Axe84] *Tit-for-Tat* Cooperates on the first move, then it does what the other played did in the previous turn. This strategy has several slightly different versions, such as the

Suspicious Tit-for-Tat which starts with **D** instead of **C**.

[Mat17] *Pavlov* is based on a similar principle, but it checks the type of payoff the player received in the previous round instead of the opponent's moves. We divide the payoff matrix into Positive and Negative outcomes. If the last round resulted in a Reward or Temptation - both positive outcomes for the player - then it will repeat the previous move. If the outcome was a Sucker's Payoff or Punishment then it switches from the previous move.

Strategies can be also probabilistic, the simplest example is the *Random* strategy - e.g. the player cooperates with a probability of 50%. We can take for example *Extortion* and *Equalizer*, both are Zero-Determinant strategies. They start with cooperation, then based on previous outcome play **C** with a probability of p. An example is proposed in [Mat17]: if the previous payoff was R, then $p = 3/4$. In case of P or S, $p = 1/4$; and $p = 1/2$ if the last move resulted in T.

[Axe84] Axelrod's first and second Prisoner's Dilemma tournament were both dominated by Tit-for-Tat, one of the simplest strategies registered for the competitions. Axelrod concluded based on the tournament results that cooperation can emerge, even amongst selfish players. In fact, our research is around similar circumstances. Given the competitive environment of our formulated problem, conflict analysis is necessary when our nodes meet in the game. One of the players will need to backtrack and give way for the other to enable progression on its path or a player might decide to share false information. These conflict situations will boil down to Cooperation or Defection and cooperation strategies, for which the study of Prisoner's Dilemma provides useful resources.

2.4 Learning Agents

In section 2.1 we introduced agents and four basic agent categories: Reflex Agents, Model-Based Agents, Goal-Based Agents and Utility-Based Agents. [Rus03] The performance of all types of agents can be improved via *learning*. Telling programs explicitly what to do and how to do it in many cases would require a vast amount of work. For tasks that are complex to program manually, building learning machines and teaching them to perform is the preferable approach. A learning agent is not limited by its initial knowledge, it can function in unknown environments and actively adopt to changes.

A learning agent is composed of four major conceptual components, as depicted by Fig. 2.9 below.

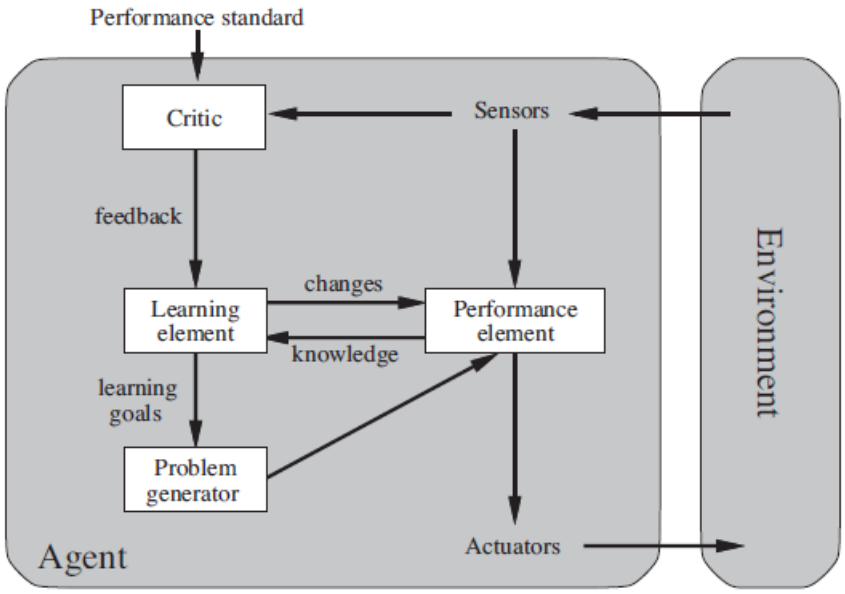

Figure 2.9: [Rus03] Learning Agent

The *performance element* processes the sensory input - the percepts - and determines an action to take via a decision making process. Essentially, the performance element component is a "general" agent by itself, the learning ability is gained through the extension components. The *learning element* is responsible for improving the performance measure of the agent, based on evaluation feedback from the *critic*. The learning mechanism used in the learning element is specific to the agent design. The critic provides its feedback by comparing the agent's behaviour to a fixed performance standard. The critic component is necessary, because the pure information gained from percepts do not provide an indication on how well the agent is doing. These components allow the agent to eventually learn which action is the most rewarding and it will organize its behaviour accordingly. However, the best action known to the agent is not guaranteed to be the optimal strategy overall. The *problem generator* component is responsible for acquiring new knowledge and experiences through exploratory actions. These experiments might lead to the discovery of better strategies, even if the agent is required to take sub-optimal actions - in comparison to its existing action set.

There is a variety of learning methods available that use different components or interpret components in an other way. However, in all cases, learning in the context of intelligent agents is the process of improving the agent's performance measure by repeatedly adjusting the agent components in attempt to achieve a better match between the feedback information and the performance standard.

[Rus03] Data can be used for learning and to improve any agent component. The methods we can use and the amount and type of improvement we can realize depends on four aspects: the chosen component, existing knowledge the agent posesses, the representation used for the data and the component and the type and amount of feedback available. Based on the components, the

agent can learn from condition - action relations, action - result relations, learn to recognize similar data - e.g. images -, and can learn how its utility function works based on feedback and results.

We can represent states in three ways:

- In *Atomic* representation, we do not know or do not care about the internal structure, we have transitions between black boxes.
- With a *Factored Representation*, we can have a variety of attributes maintained as a vector or list - Booleans, real values or values from a fixed set of symbols.
- With *Structured Representation* states are composed of individual objects that have their own set of attributes and the relationships between them are also defined.

Based on prior knowledge, we can carry out *inductive learning* by deriving a general function from input - output pairs. Taking a known general rule, the agent can deduce a new logically entailed rule - that allows for more efficient processing - using *analytical* or *deductive* learning.

[Rus03] Based on the available feedback we classify machine learning into three main types. If the agent can observe input - output or action - result relations , *Supervised Learning* can be implemented. The agent can learn which actions map to what consequences. Given a set of input - output pairs $(x_1, y_1), ..., (x_n, y_n)$, where every output y_n is generated from the corresponding input x_n using some unknown function $y = f(x)$. Our task is to generate a hypothesis function h, which can estimate f. The known set of input - output pairs is the *training set* and a distinct set is used to test the accuracy of h, the *test set*. Depending on the type of the output we group supervised learning tasks into two main categories. If y is a number, we are dealing with a regression problem. If the output is generated from a finite set of values - such as the set of colours - then we have to solve a classification problem. Within Supervised Learning, a wide variety of algorithms exist. Some of the most popular ones are the K-nearest neighbour algorithm, linear regression, support vector machines, neural networks and decision trees.

With *Unsupervised learning* there is no explicit feadback available, the agent must learn by recognizing patterns in the percepts. [HS99] Unsupervised learning can be summed up as supervised learning without information on the output. One of the approaches to solve this problem is to discover clusters in the data. [RD09] With *cluster analysis* we attempt to discover similarities and separate the unlabeled dataset into a discrete set of data structures. The unsupervised problem is subjective in the sense that the resulting clusters heavily depend on the technique and parameters used. There is no universally accepted formal definition of cluster, however a common property that must hold in ever case, is that members of the same clusters must be similar and members of different clusters must be dissimilar to each other. These similarities and dissimilarities must be defined in a clear way.

In the case of *Semi-supervised learning*, we have relatively few labeled examples - whose labels may or may not be as useful or informative as we would expect - and we are trying to predict the labels of a large set of unlabeled data. Semi-supervised learning is used, when a significant

amount noise is present in the labels and the number of elements in the training set is low.

The agent can also learn by receiving performance feedback in the form of rewards or punishments for the actions it takes. Naturally, it is desirable for the agent to maximize its rewards and minimize the punishments, so gradually it will learn to choose moves with better payoffs. This method of learning is called *Reinforcement Learning*.

2.4.1 Reinforcement Learning

We discuss reinforcement learning in more detail, as this is the approach we will implement to investigate learning in our proposed environment. [SB17] With reinforcement learning the agent must explore the available actions and learn which ones to perform in order to maximize the reward. This property poses a challenge, which is not present in supervise and unsupervised learning problems. An agent has to perform actions that it has already experienced to be rewarding to maximize its efficiency. However, the agent also has to explore other - perhaps suboptimal - action to allow it to make better choices in the future. The dilemma exists because the agent cannot be successful by exploiting or exploring only, it must do both. The trade-off and the balance between exploitation and exploration remains unresolved, despite the intensive research by mathematicians for the past decades. Unlike with other learning approaches reinforcement learning considers the problem of the goal-based agent in an unknown environment as a whole, instead of approaching it as a composition of several sub-problems.

[SB17] There are four fundamental components required to define a reinforcement learning system:

The *policy* determines the behaviour of the agent. The policy is a function that maps percepts to actions. In the context of games the policy would be the strategy of the player at a given time.

The *reward signal* is an immediate feedback from the environment indicating the performance of the last move the agent's policy selected based on the state it reached. The reward signal defines the goal of the agent and is the primary basis for changing the policy.

While the rewards describe immediate effects, the *value function* describes what is desirable in the long run. The value of a state s defines the maximum amount of reward the agent can expect to accumulate, starting from s. It is possible that a high value state's expected maximum payoff is reached through low reward policies, while the agent could only accumulate a lower amount of utility through the high reward policies.

In order to implement planning in the system we need a *model* that describes the environment. The model allows the agent to infer prediction about the changes in the environment - given a state, the model might predict what the resulting environmental state would be if the agent choose to perform a certain action. The model might also estimate the reward in the expected next state. Model-free methods omit planning and rely explicitly on trial-and-error based learning.

Although applications exist where the value function can be omitted - e.g. evolutionary methods -, the main objective in reinforcement learning is to accurately estimate the value function. Depending on the complexity and dimension of the state and action space we classify approaches into two main categories: *Tabular Solution Methods* and *Approximate Solution Methods*.

2.4.2 Tabular Solution Methods

[SB17] Tabular solution methods can be used in scenarios, where the state and action space are small enough - i.e. finite - for the value functions to be represented as tables. Tabular methods can find exact solutions,thus they can discover the optimal value functions and the optimal policies.

2.4.3 Approximate Solution Methods

no exact solution, just aproximation, infinite states, need to generalize

Chapter 3

Implementation

This chapter introduces the implementation of the theoretical concepts discussed in the previous chapter, discussing the development steps and the decisions made.

3.1 Definition of the game

Our uncertain environment is represented by a maze. Robots or Nodes - represented by intelligent agents - are placed within the unknown environment with the task of finding a target cell.

3.2 Multi-agent framework

3.2.1 System overview

3.2.2 The Nodes

3.2.3 Node Interaction

Movement

Communication

3.3 Strategies

3.4 Learning Agents

Chapter 4

Experiments and Results

4.1 Programmed behaviour

4.1.1 Evaluation of strategies

4.1.2 Emerging trends

4.2 Learning Node

4.3 Evaluation

4.4 Emerging behaviours

Chapter 5

Conclusion

Bibliography

[SB17] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. Cambridge, MA : The MIT Press, 2017.

[Mye91] R. B. Myerson. Game Theory : Analysis of Conflict. Cambridge, Mass: Harvard UP, 1991.

[Bar49] E. N. Barron. Game Theory : An Introduction. Second ed. Hoboken, N.J.: John Wiley and Sons, 2013.

[Von44] J. Von Neumann and O. Morgenstern. Theory of Games and Economic Behavior 60th Anniversary ed. Princeton: Princeton UP, 2007.

[May82] M. Smith. Evolution and the Theory of Games Cambridge: Cambridge UP, 1982

[Ras07] E. Rasmusen. Games and Information : An Introduction to Game Theory 4th ed. Oxford: Blackwell, 2007.

[Axe84] R. M. Axelrod. The Evolution of Cooperation New York: Basic, 1984.

[Nas51] J. Nash. "Non-Cooperative Games". The Annals of Mathematics. Vol. 54. pp. 286 - 295. September 1951.

[Mat17] P. Mathieua and J. Delahayeb. "New Winning Strategies for the Iterated Prisoner's Dilemma". Journal of Artificial Societies and Social Simulation. Vol. 20. pp. 12. October 2017.

[Nwa96] H. S. Nwana. "Software Agents: An Overview". The Knowledge Engineering Review. Vol. 11:3 pp. 205-244. 1996.

[Rus03] S. J. Russel. Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall, 2003.

[Woo02] M. J. Wooldridge. An Introduction to MultiAgent Systems Chichester: Wiley, 2002.

[Sea69] J. Searle. Speech Acts: An Essay in the Philosophy of Language. Cambridge: Cambridge University Press, 1969.

[Sta00] K. Stathis. "A Game-based Architecture for Developing Interactive Components in Computational Logic". The Journal of Functional and Logic Programming. Vol. 2000. Article 5. March 2000.

[SS96] K. Stathis and M. J. Sergot. "Games as a metaphor for interactive systems." People and Computers XI. pp. 19-33. August 1996.

[HS99] G. E. Hinton and T. J. Sejnowski. Unsupervised Learning: Foundations of neural computation MA,Cambridge : MIT Press, 1999.

[RD09] R. Xu and D. C. Wunsch. Clustering. Oxford: Wiley, 2009.

Appendix A

Other appendices, e.g. code listing