ARTIFICIAL INTELLIGENCE

COMP1694

UNIVERSITY OF GREENWICH

Table of Contents

Part A

1. Assume that the universe of discourse is the set of people studying or working at the University of Greenwich. Rewrite the following statements in the form of predicate logic.

    a. Each person is either a student or a staff.

    ∀x(Student(x) ∨ Staff(x))

    b. Each lecturer teaches some courses.

    ∀x(Lecturer(x)→(Teaches(x,y)~Courses(y))

    c. Some hard-working people are not boring.

    ∃x(Hard-working(x) ^¬Boring(x))

    d. Hard-working people are respectable.

    ∀x(Hard-working(x) → Respectable(x))

    e. Everyone knows some hard-working people.

    ∀x ∃y (Knows(x,y) ^ Hard-working(y))

[10 marks]

2. (a) Use truth table to verify the following equivalence:

A → B ≅ ~A ∨ B

A	B	**A → B**	~A	B	**~A ∨ B**	**A→B ≅ ~A ∨ B**
T	T	**T**	F	T	**T**	**T**
T	F	**F**	F	F	**F**	**F**
F	T	**T**	T	T	**T**	**T**
F	F	**T**	T	F	**T**	**T**

and use your own example to explain your understanding of the actual meaning of "A materially implies B" in propositional logic.

A (materially) implies B is denoted as A→ B. This means when A is true, and B is false, the outcome will be false. Any other outcome of A and B will be true. A truth table above and below shows what A materially implies B.

P	**Q**	**P→Q**
T	T	T
T	F	F
F	T	T
F	F	T

An example can be **"it is morning, I can't see the moon".** We have four possibilities according to this truth table.

Example is shown below.

1. It is morning (T) and I can't see the moon(T). The outcome is (T).
2. It is not morning (F) and it is cloudy, and I can't see the moon (F). This statement makes

sense. So according to P→Q, this outcome is T.

3. It is morning (T), but I can see the moon (F). The outcome was (F).
4. It is not morning (F), but I can't see the moon as it has not come out yet (T). The outcome is (T).

[8 marks]

(b) Distinguish between *deductive*, *inductive* and *abductive* reasoning; give an example of the appropriate use of each.

Deduction reasoning is when you derive the logical conclusion of the necessary conclusion. However, abduction reasoning is the process of seeking the reason from a state that has already occurred. **Abduction** is a "best guess" from inferring from the statement. **Induction** is a generalisation from inferring cases to cases. The difference between all three is that deductive is that it is guaranteed to be true, whereas Inductive is that it is probably to be true. On the other hand, abductive is a "best guess". Each example of deductive, inductive and abductive reasoning is shown below of the difference of each. We learn the different approaches of each of them. Each of them have a different result. Examples is shown below.

Abductive:

Suppose I had a laptop and the Help Desk has many laptops. You may infer from is the laptop was borrowed from the Help Desk.

Deduction:

All gamers like games. Therefore, Matthew is a gamer. We conclude from this is that Matthew likes games.

Induction:

My brother likes apples. My father likes apples. I like apples. Therefore, everyone related to me likes apples.

[9 marks]

(c) List the four main representation schemas learnt from this course and give a typical example(s) for each of them.

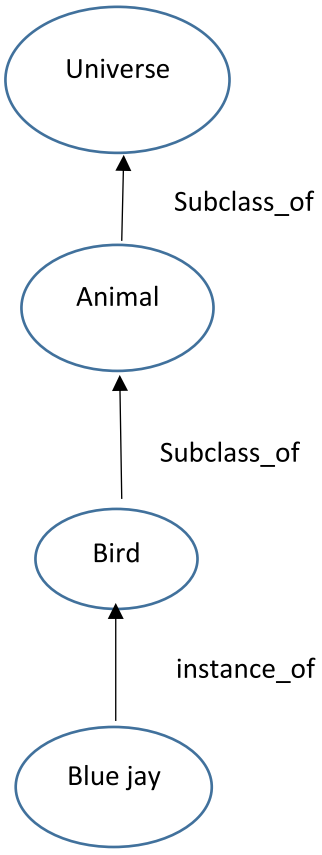
These are the four main representation schemas that I have learnt over this course. Beside them is a type of representation schemas learnt from the course and a typical example will be named.

- **Logical Representation**

Predicate Logic and Propositional Logic are two examples of Logical Representation. **Predicate Logic** is a language that expresses expressions and statements into a predicate logic**:** An expression for predicate logic is, "Everyone loves everyone". This will be expressed as, **"∀x ∀y (Loves(x, y))".**

- **Network Representation**

Semantic Network is an example for Network Representation. A semantic network is a graph representation that shows objects and arcs as a relationship. An arc represents a relationship. In this case, the "subclass_of" is the arc of the representation. Each node that is within the relationship shows an object. A "instance_of" shows the indication that it is used within the class. For example, below is a semantic network that shows that the class instance from Blue Jay to Bird is a member of a bird. In other words, Blue Jay is a type of Bird.



- **Structured Representation**

Frames is a type of Structured Representation. Frames is a packet of information that uses an object for it. For Frame, it contains frame name, slots, facets and fillers. An example is shown below for Frames. Frames can be adapted and also made into much more detail. However, the example shows how the frame works.

Instance_of	**facet**	**Filler**
	VALUE	Lecturer
Name	**facet**	**Filler**
	VALUE	Dr._Martin
Course_teach	**facet**	**Filler**
	VALUE	Programming_Components
Room	**facet**	**Filler**
	VALUE	KW215
Level	**facet**	**Filler**
	VALUE	4

- **Procedural Representation**

Production System is a type of Procedural Representation. A production system has a set of rules that it needs to follow. A database of facts, a set of rules and interpreter. This works by using IF and ELSE statements to answer the question the user is trying to make. A scenario that can be used if a person is qualified to purchase an 18+ game. The database will have the following code to give a YES or NO answer based on the ID.

```
R1. IF<show id? Is 18+>
THEN
read YES.
clear database
finish.

R2. IF<show id? Is 18->
THEN
read NO.
clear database
finish.

R3. IF<no show id?
THEN
read NO.
clear database
finish.
```

[8 marks]

3. (a) Express the following knowledge as a Prolog rule(s):

X is Z's grandparent if X is Y's father or mother, and Y is Z's father or mother.

grandparent(X, Z):- (father(X,Y); mother(Y,Z)), (father(X,Y); mother(Y,Z)).

[5 marks]

(b) Consider the following Prolog program/database:

```
a(X):- b(X), c(X), d(X).
a(X):- c(X), d(X).
a(X):- d(X).
b(1).
b(a).
b(2).
b(3).
d(10).
d(11).
c(3).
c(4).
```

Given the query? - a(X). What are the successive variable bindings that the variable X gets when the above query is run. Separate bindings by a comma (i.e., 1, a, 5, ...).

We consider the variable bindings that the above query runs cannot be run in the order that is set. To run the query successively, the variable bindings need to be to run. The query runs:

1, a, 2, 3, 3, 4, 10, 11.

These bindings above runs in alphabetically. First runs all is b(1), b(a), b(2) and b(3). Later, C(3) and C(4) runs. Alongside this D is the last to run is D(10) and D(11). Below shows how these in order is executed. Referring to Figure 1, it shows which variable bindings are successful and which variable bindings are not.

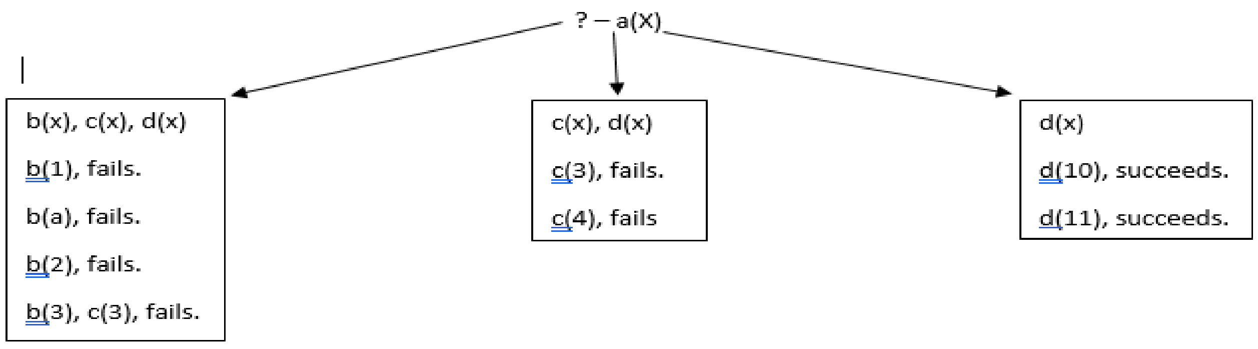
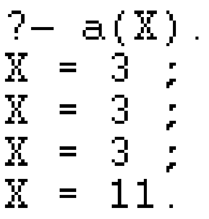


Figure 1 shows how the search is explained and what executes and what does not.

Referring to Figure 1, this shows the approach to how Search in Prolog is done. The 'fails' show that the same of each b(1), c(1), and d(1) cannot be found. Therefore, it fails. Each time this fails, it goes to the next variable and tries for it to be successful like this. The answer to this when the query runs is d(10), and d(11). Another similar example is when d(10) can be changed to d(3). The outcome below is a screenshot in Prolog that it is changed, and the outcome is displayed. The answer is displayed and the outcome is from b(3), c(3) and d(3) when changed. b



```
?- a(X).
X = 3 ;
X = 3 ;
X = 3 ;
X = 11.
```

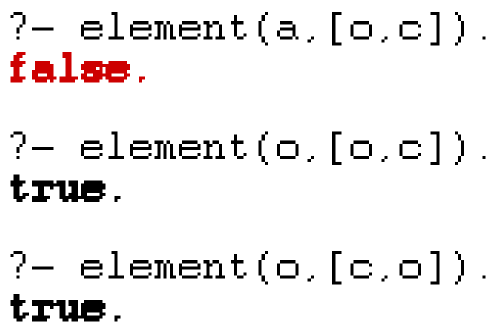
[8 marks]

(c) Explain what the following program does:

Line 1: element(Element, [Element|Tail]) :- !.

Line 2: element(Element, [Head|Tail]) :- element(Element, Tail).

The example above shows it trying to find one element. Once it finds one member, it stops backtracking. It stops backtracking because of the "cut". The exclamation mark "!" is known as the cut which prevents any other goal to be executed and fixates all the variables to be followed towards the left. If it does not find one element, it uses recursion, goes to the Line 2, and finds another element. It uses the list to find the first element. Referring to Figure 2, it shows in Prolog the difference between what the program does. This is trying to find the element in the following list. Once it has not found it, it says this statement is false. The second element is put in as 'o' and it has been put in as 'o' in the list. All the program does is find the exact



```
?- element(a,[o,c]).
false.

?- element(o,[o,c]).
true.

?- element(o,[c,o]).
true.
```

Figure 2 shows the execution of the statement.

match as its first one. Once it finds it, it stops backtracking and stops executing. Once it is put in the list, the cut shows that it has been stopped backtracking and it does not continue.

For the statement above, the statements are an empty list. Referring to Figure 2, therefore, when the statement is executed, the statement is not empty.

[6 marks]

(d) What is the actual meaning of "no"s in Prolog? Explain the so-called "*negation as failure*" strategy used in Prolog.

In Prolog, "no"s mean it **is impossible to prove**. At first, one would assume that it means false. However, it is quite different, and it shows that it does not work at all and it cannot be proven.

Negation as failure is used to represent facts and rules. Any facts that is known in the database can be represented and executed as true. However, if the fact is not in the database, the outcome of this would be false. Any unfinished statement could cause problems, as it needs to be finished.

Referring to Figure 3 and Figure 4, it shows an example of the difference of what is included within the database. When typed, "greenwich(X)." this shows all of the possibilities of what is included within the database. When typed in the different possibilities, it shows the outcome as mentioned above.

[6 marks]

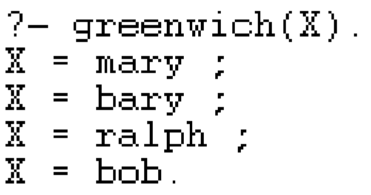
greenwich(X): - staff(X); student(X).

staff(mary).

staff(bary).

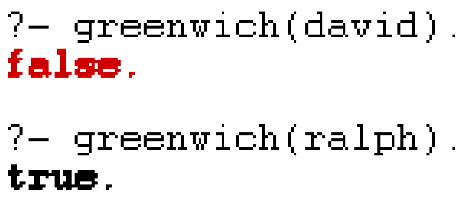
student(ralph).

student(bob).



```
?- greenwich(X).
X = mary ;
X = bary ;
X = ralph ;
X = bob.
```

Figure 3 shows the outcome of the statement in Prolog



```
?- greenwich(david).
false.

?- greenwich(ralph).
true.
```

Figure 4 shows the outcome of the statement in Prolog

Part B:

4. Write a research report on temporal logic and/or their application in the domain of Artificial Intelligence

Introduction

This part of the report is a literature review, where discussions are found for the study of how temporal database management relates to the temporal logic, or “time” in Artificial Intelligence. The aim of this report is to discuss the different types of databases and how they relate to *time*. These topics will be discussed in depth and it will together be included with my understanding, observation, critical analysis and evaluation of these topics. Furthermore, these discussions will be included by knowledge gathered by existing research of professionals and academics.

Temporal Database Management

A temporal database is a database that “records *time-varying information”*. Most database applications are temporal in use e.g. temporal database is within use is medical records, accounting and banking (Jensen and Snodgrass, 1999). Medical records the patient’s history and this is important to the doctor to access the current mental state of the patient. The doctor needs to know not only her current state, but by checking the history of the situation, including: How long has the patient been ill? Has this previously occurred before? Have you got any allergy to taking medicine? (Ma, 2017). This is important for the doctor to access the situation and give the patient the right medicine according to her records. The same principle for medical records applies to the given suggestions that relate to temporal database such as banking, and accounting. As temporal database requires time, it has aspects that usually include *valid time* and *transaction time* (Edelweiss et al., n.d.). These two-different types that usually are included that will be explained at a later stage in the report.

Valid Time

Valid time is used within temporal database that requires “*attribute values were true reality”* (Snodgrass, 1992). Valid time is usually supplied by the user and these values should be true in reality. Valid time may be in the past, current or occur in the future. Future times can be in the future if known it is true (Snodgrass, 1992). Suppose I use a scenario of having an appointment. The first action I make is the availability of the appointment. This could be from January 1st, 2018 to December 31st, 2018. We are ready to query these dates, and this uses transaction time, which is example later on in the report. These values are only true in reality.

Transaction Time

Unlike valid time, transaction time enables "*concerns the time the fact was present in the database as stored data"* (Snodgrass, 1992). We all know that databases can enable the user to store information, create tables and deleted information. Transaction time in which the object is being stored in the database, is sometimes called *recording time* (Ma, 2017). We can use the previous example within valid time about making an appointment. Suppose I would like to have an appointment at 14th May 2018. I query this, and this would be in the *future* that I an event is going to happen. This query would be set in and be inserted within the database alongside the time. This could be deleted, because if I am busy this day. It is flexible and stores the time within the database. This makes it so that it starts from the date put in to infinity.

Types of Databases within Temporal Database Management

Historical Databases

Historical database shows the past data and uses valid time to view these databases. As Monica Wachowicz states in her book, historical database "*provide the knowledge about the past by supporting valid time (the time data have changed in the real world" (Wachowicz, 2003).* Historical database comes in use to those companies where they would like to look back at records. However, this is a *type of database* that can be used. For example, a business that notes down their sale targets; they look at their database and check their previous history of how their sale targets have done over the last year. They can then compare and make additional changes to improve the current state of the sales.

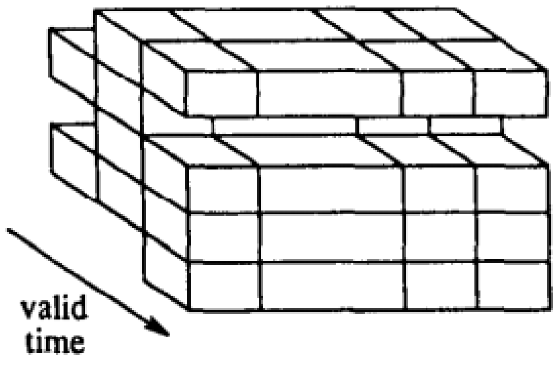


Figure 5 shows a representation of historical database.

Referring to Figure 5, it shows the representation of how historical database is viewed within a database (Snodgrass, 1987). It shows how it shows all the data within the database and how it uses valid time to view this. This information can be updated for record purposes. Once this information is updated, the previous information cannot be retained, or updated. You can only update the current information that the valid time provides historical data.

Rollback Databases

Rollback Databases expresses and uses transaction time only. The term *rollback* comes from selecting a snapshot state. By putting this into a database, it comes up with a rollback database. Referring to Monica Wachowicz, she also states that "*Rollback databases provide a sequence of snapshot states of the database by offering support for transaction time (the time the data are stored in the database"* (Wachowicz, 2003). A snapshot state is a "*state or an instance of a database in is current contents"* (Snodgrass, 1993). This goes well together with transaction time as Figure 6 shows the representation of snapshot state indexed by transaction time (Snodgrass, 1987). As we go right, a new block of rollback database is added to the database. Similar to historical database, changes that are made, they are only made to the most recent in the database. The disadvantage to making changes to the current state is that changes can only be made to the current state. Therefore, if there were errors within the database that the user has noticed, they are unable to change the errors they have found in the previous data. They can only update it and show the updated version of it (Ma, 2017).

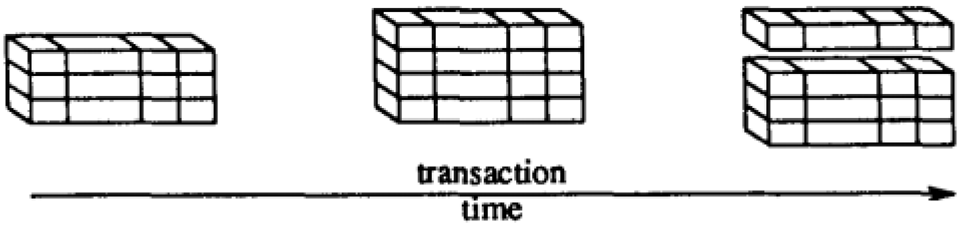


Figure 6 shows a representation of rollback database.

Bi-Temporal Databases

Bi-Temporal Database is different to historical database and rollback database as bi-temporal database uses both *valid time* and *transaction time*. This provides both of **historical** and **rollback** information that is in use. Referring to Figure 7, it shows a representation of a bitemporal database. Between all of this, t(1) and i(1) is a transaction time that is valid; ht2 and ht3 is in the history timescale (Kumar, Tsotras and Faloutsos, 1998).

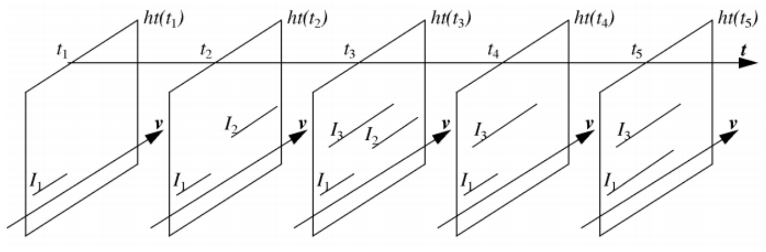
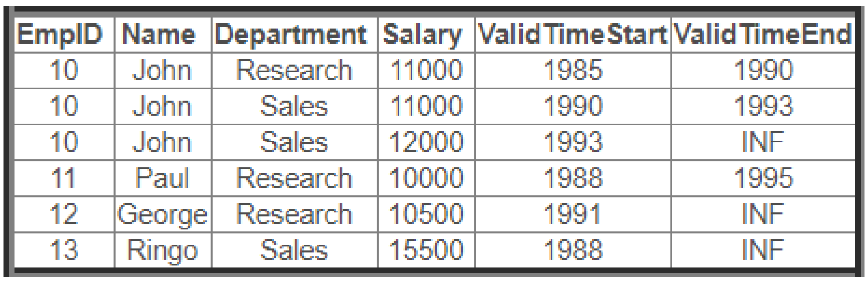


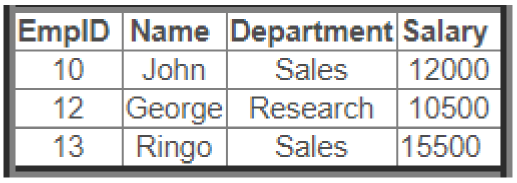
Figure 7 shows a conceptual view of bitemporal data

Temporal Database vs Non-Temporal Database

All these databases that have been mentioned all require time values for their respected databases. All of them are different to each other, but have the same principle. However, we have different types of databases that are not temporal. Non-temporal databases are structured in a form of DBMS and an example of the use of it is Oracle. Referring to Figure 8, the two databases show the difference between a temporal database and a non-temporal database. As you can see, the main difference between the two is essentially the time (TimeConsult, 2005). However, for the non-temporal database, the same ID can be updated. However, the Temporal Database must create another ID that is the same and update the values within it. Here we have shown that John has moved departments from Research to Sales and alongside this has his salary upgraded. Once this is updated, 'INF' is inserted as an infinity. This can be later updated.



EmpID	Name	Department	Salary	ValidTimeStart	ValidTimeEnd
10	John	Research	11000	1985	1990
10	John	Sales	11000	1990	1993
10	John	Sales	12000	1993	INF
11	Paul	Research	10000	1988	1995
12	George	Research	10500	1991	INF
13	Ringo	Sales	15500	1988	INF



EmpID	Name	Department	Salary
10	John	Sales	12000
12	George	Research	10500
13	Ringo	Sales	15500

Figure 8 shows the difference between temporal and non-temporal databases.

Temporal Database Schemas

Schemas in databases are a blueprint of how the database is constructed. Any database has a schema that they follow. This is the same for temporal database too. Temporal database schema is represented as "$TR(T_{transaction}, A_1, \ldots, A_n, T_{valid})$" (Ma, 2017). These are known as **T**transaction, **A**argument, **T**valid, and meets(**T**firstargument, and **T**secondargument) (Ma, 2017). These are all represented within the database as a schema in temporal.

Case Study

A case study will be presented below to show how temporal database is used and how it will be recorded.

A patient came in, named John, stating he has had a headache for a few weeks. Dr Matthew checked this and prescribed the patient, John, pain-killers (drug b). John had taken some previous drugs that the doctor ordered to take the pain-killers he has given to feel better. The doctor updated the records to put into the system and booked him another appointment to see his condition if it gets worse. Three days later, John came in with a worse condition and Dr Sully prescribed John with another drug antibiotics (drug c) and Dr Sully updated his records. A few days later, his condition improved and felt better.

Representation of the database in Temporal Database Management

Table 1 shows a representation of how temporal database is stored.

T(trans)	Patient	Doctor	Drug	Status	T(valid)
I(t1)	John	NULL	NULL	pain	i(v1)
I(t1)	John	John	a	worse	i(v2)
I(t1)	John	Matthew	b	NULL	i(v3)
I(t2)	John	NULL	NULL	pain	i(v1)
I(t2)	John	John	a	worse	i(v2)
I(t2)	John	Major	b	better	i(v3)
I(t2)	John	Sully	c	NULL	i(v4)
I(t2)	John	NULL	NULL	better	i(v5)

Tabel 1 shows a representation of how the case above is represented. It shows that temporal databases are takes much more effort to update the system. The case and table has been adapted from Jixin Ma's Temporal Representation and Reasoning in Artificial Intelligence (Ma, 2017).

5. Literature Review applying Artificial Intelligence techniques such as learning, search and planning to games

Introduction

In this report, I will be going to writing a report on Artificial Intelligence techniques such as learning, search and planning to games. I will be focusing on past and recent applications alongside future research for Artificial Intelligence and games. I will be applying various techniques that is applied to games. Alongside this, I will be stating how these techniques will be reflected as an example that accurately reflects real life problems for this.

History on Artificial Intelligence games

When Artificial Intelligence in gaming started off, it was simple. In 1968, it started off with "Tennis for Two", which was used and played on an oscilloscope. James Wexler mentions in his report how "Pong" was one of the earliest versions of it. However, these versions later got improved to games such as "Kung Foo", "Mortal Kombat" adapted from (Wexler, 2017).

In 2007, AiGamesDev launched an annual games award for recognition to Artificial Intelligence games. In 2009, the overall Best Combat in AI was awarded to Killzone 2 (Parker, 2010). Killzone 2 is a first-person shooter video game that was developed for PlayStation 3 which was published by Sony. Killzone 2 uses AI and has various combat techniques that they used to win this award. Some Artificial Intelligence techniques that have been included within the game are movement, behaviour and influence map. These techniques are the foundation of what they have used and have made these techniques advanced to get their game to do what they want today. More of these techniques that have been mentioned would be discussed at a later stage within this report.

Open problems on games for Artificial Intelligence

Issues that occur in games for Artificial Intelligence is the demand of 'bots' behaving human-like. Bot is a character within a game that is controlled by the computer. One of the main problems for developers is to evaluate 'human-like intelligence' means for a bot. This is one of the main problems, a hard problem, in gaming (Lara-Cabrera and Nogueira-Collazo and Cotta and Fernández-Leiva, 2017). However, one of the problems that users constantly face are glitches within the game. This is when the 'bots' do not follow the

Figure 9 shows a glitch within a game that uses Artificial Intelligence.

Artificial Intelligence code that is originally set for them to do and they do something that does not follow the specification that is originally set. Glitches prevent you from playing the game that is originally set. Referring to Figure 9, it demonstrates an example of problems that occur within Artificial Intelligence gaming. This shows a 'bot' running into the sky and is unable to shoot the bot. This does not follow thee AI that is originally set from the start and is an example of glitches within AI game.

Future Research on games for Artificial Intelligence

Unlocking new possibilities for future research in Artificial Intelligence for games is to aim to develop making NPCs more powerful for the current efficiently to stay intact. NPC stands for Non-Player Characters that are not controlled by the player playing. Jeffery and Christopher states about NPCs, "*include all the inhabitants of the game world who aren't being played by human beings*" (Georgeson and Child, 2017).

The main aim of the development will be for the user experience to be enhanced. Virtual Reality enhances user experience by visualising and playing the game virtually. This is one of the experiences currently that the world alight. Andrew Wilson, CEO of Electronic Arts, states, "*Your life will be a video game*" (Lou, 2017). Considering Virtual Reality as a takeover, will Andrew Wilson's statement come true? Considering user experience will be developed, and Virtual Reality is a pathway to enhance on; it will be a good combination alongside the future games that will be coming out e.g. FIFA 19. With more and more extensive research for Artificial Intelligence in gaming, it has a promising future for sure.

Resources available to people who would like to work in this space

They are tons of resources for people who would like to work in Artificial Intelligence gaming. You can start by reading books on the approach to Artificial Intelligence for Games. A highly recommended book by experience users is **Artificial Intelligence for Games by** Ian Millington and John Funge Another book that is recommended is **Procedural Generation in Game Design** by Tanya X. Short and Tarn Adams. Both of these books are recommended on AI and Games site (AI and Games, 2017).

More than reading, a highly suggested tool is to practice too. Reading about these techniques is good for information, but you would like to test yourself. Graphic programming tools such as Unity is a good start to get experience on programming. A suggested tool is for users who would like to work in the field is to have a strong background in Python. Having this tool uses the AI-side of programming within Artificial Intelligence field.

By putting these tools and resources to test, it should give you the basics to start of the career within Artificial Intelligence games.

Techniques

A* Pathfinding

A* Pathfinding “*generally refers to find the shortest route between two end points”* (Cui and Shi, 2011). Another common term that is used in Computer Science is A* Search. A* Pathfinding uses a algorithm to execute the “*cheapest path*” (Daylamani-Zad, 2017). Some examples that this can be referred to is using a maze game to find the shortest path from where the user has started to where the user wants to finish. Some benefits of A* Pathfinding is that it is fast, widely used within gaming industries (Daylamani-Zad, 2017). An example that A* Pathfinding is used is on Antimodel where it shows how A* Pathfinding is efficient and fast in searching for the quickest route between two points.

How does A* Pathfinding work?

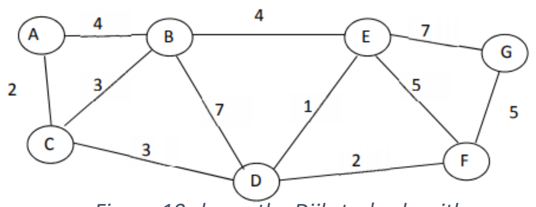


Figure 10 shows the Dijkstra's algorithm.

Using Dijkstra’s algorithm as an example, it too finds a way for the shortest path. Referring to Figure 10, it demonstrates a way of finding the answer (Dynamic Learning, 2017). The question is find the quickest route from A to G. The answer is A, C, D, F, G. This is like how A* Pathfinding works. It calculates the same and figures the cheapest route for the points shown.

A* Pathfinding is not only used in games, but can be flexible and be used in different aspects of Artificial Intelligence too. An example is when a robot is trying to find a pathway to its destination, it can be used (Adapted from Cui and Shi, 2011). All of these points made about A* Pathfinding, this will be discussed in detail of how it works at a later stage in the report.

Machine Learning using Genetic Algorithm

Machine Learning is a type of learning technique that evaluates past actions that can be used for the future. This technique is usually show promise, but it is slow (Daylamani-Zad, 2017). Machine learning starts off with a task, investigation and a possible algorithm for the solution (Carbonell and Michalski and Mitchell, 1983). These steps are the basics of Machine Learning. One example of Machine Learning that uses these steps are Genetic Algorithm.

Genetic Algorithm is one of the most promising algorithms. Genetic Algorithm is a heuristic approach that is inspired by Darwin’s theory. The selection theory is based upon a combination of natural selection and natural genetics. They combine to survival of the fittest among structures that exchange and form a search algorithm (Goldberg, 2012). The parameters of the search are formed as the chromosomes, and a collection of such strings is called population. The population is selected by the survival of the fittest. *Crossover* and *mutation* are methods that are used to select the "*survival of the fittest*" (Maulik and Bandyopadhyay, 2000). This links in with the natural genetics that was stated by David Goldberg.

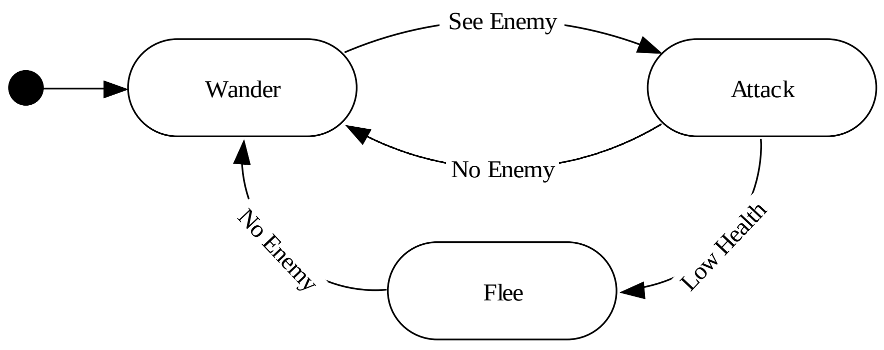
Although Genetic Algorithm is a heuristic approach to search for the best possible answer, it solves problems for various problems out there. For example, scheduling is a type of issue that can be solved using Genetic Algorithm.

How does Genetic Algorithm relate to games in Artificial Intelligence?

Genetic Algorithm can be used to tune different aspects of the game. Choosing a set of parameters to tune the bot to be fast and efficient is a way to tweaking it using this technique. Using good set of rules is the main reason for using Genetic Algorithm to "*tune the parameters as well as a human*" (Cole and Louis and Miles, 2004). Some points that Damon Daylamani-Zad, includes within his lecture about games in Artificial Intelligence, states that using Genetic Algorithm is often expensive and it is suggested to try it offline. It is also not well understood and may not give the best answer according to the AI settings (Daylamani-Zad, 2017).

Finite State Machine

Finite State Machine is a "*well-suited model for describing a synchronous sequential machine*" (Cheng and Krishnakumar, 1993). Finite State machine can be used to represent in games by using a set of states. Finite State Machine is often similar to State Diagrams used in UML. A set of behaviours that is used to create a finite state machine. Below is an example of how a player is going to attack an enemy. It shows us that if the player sees an enemy, it attacks it. However, if it is low health, or no enemy, it runs and wanders for another enemy. This representation shows us in a few diagrams how Finite State Machine works. This can be used within other situations for diagrams and programs (Daylamani-Zad, 2017).



MinMax

MinMax is at the heart of *almost* every computer game. Some of these games that applies this logic of MinMax is the Chess, Checkers, Monopoly, and many other board games (Daylamani-Zad, 2017). Referring to Figure 11, it shows how a search tree works (Ntu, 2017). The picture shows below of how the logic applies. This is a search tree as an example of how it is used. First, for each of the two blocks, it chooses the minimum of each of them. It shows that for the first row. Then, Max has a turn and picks the Maximum out of the two. It goes in this pattern of Minimum and Maximum out of the two. Once it goes to one logical move, that means that this is the best move. However, the logical sense for using this numbers can be used in other realistic board games too.

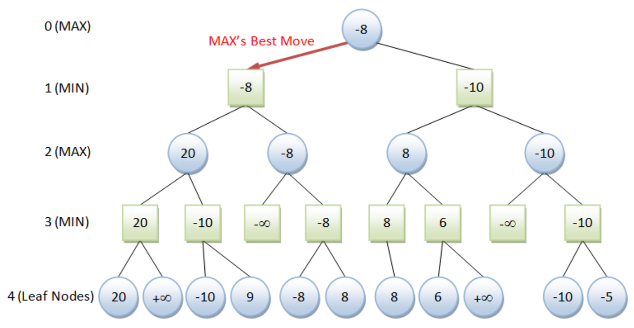
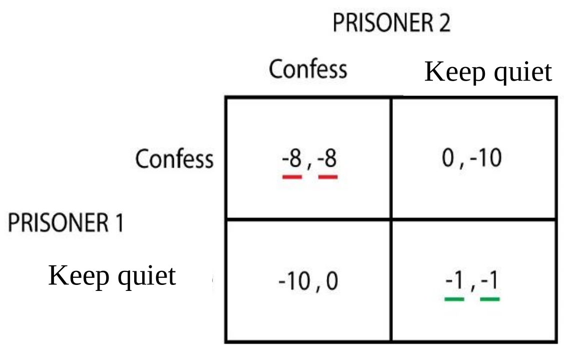


Figure 11 shows a search tree for MinMax.

Prisoner's Dilemma

Prisoner's Dilemma is a game theory that has a principle that shows two individuals might not make the logical answer when given a scenario. Referring to Figure 12, it shows a common scenario where two prisoners must confess, or keep quiet. The logical sense is for both to keep quiet and hope the other prisoner keeps quiet too. If the other confesses, the one who kept quiet would have 10 years in prison.



		PRISONER 2	
		Confess	Keep quiet
PRISONER 1	Confess	-8 , -8	0 , -10
	Keep quiet	-10 , 0	-1 , -1

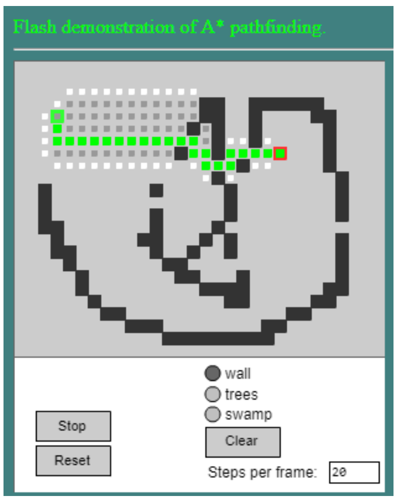
Figure 12 shows the consequences of prisoner's dilemma.

Game examples that accurately reflect real life problems

Maze and Drawing Path

A* Pathfinding is explained clearly above of how it works. Below are two examples of how the techniques is related to real-life problems. One is drawing your own shape and finding the quickest path from A to B. The other is trying to solve a maze using A* Pathfinding.

Referring to Figure 13, it demonstrates how two points is clearly found the easiest route between each other. As soon as 'Find Path' is clicked, a path is drawn around Point A to find the nearest of its Point B. It draws out the green line as a mark to its destination.

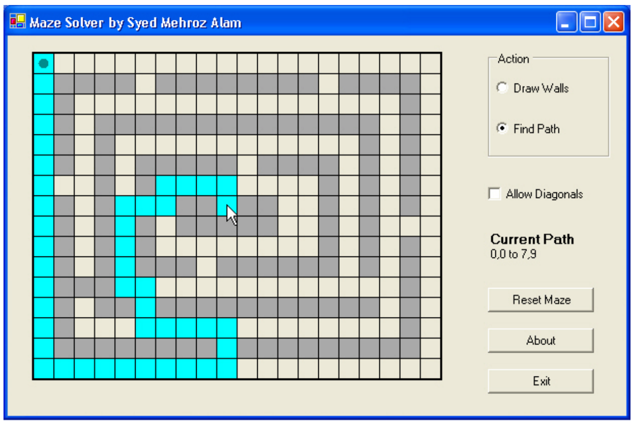


Steps per frame: 20

Figure 13 shows how A* pathfinding

Once the ‘grey area’ is complete, the path is complete. The option is for the user to draw any random shape as you can. This enables the complexity of A* Pathfinding is very advanced. Figure 13 can demonstrate two locations and its quickest route. This can be used as a real-life problem as one is trying

Another example of where A* Pathfinding is where it can be used is within a maze. Referring to Figure 14, it shows how similar Figure 13 is with Figure 14 (Alam, 2005). They both work the same, but for Figure 7, the Maze is set in stone. It doesn’t change and allows the easiest and cheapest path to occur.


Maze Solver by Syed Mehroz Alam

Action
Draw Walls [✓ Find Path]

Allow Diagonals

Current Path
0,0 to 7,9

Figure 14 shows a maze solver using A Pathfinding.*

Counterstrike

Genetic Algorithm has been used in games, and a first-person shooting game was applied in it. The game that Genetic Algorithm was used is called Counterstrike (Watson and Azhar and Chuyang and Pan and Chen, 2008). Referring to Figure 15, it demonstrates the quality of how the first-person shooting game was used (Counter Strike, 2017). This is an accurately real-life describes the graphics and artificial intelligence represents in counterstrike.

Figure 15 shows the gameplay of Counter Strike

References

AI and Games. (2017). *Recommended Resources*. [online] Available at: http://aiandgames.com/recommended/ [Accessed 12 Dec. 2017].

Alam, S. (2005). *Maze Solver (shortest path finder) - CodeProject*. [online] Codeproject.com. Available at: https://www.codeproject.com/Articles/9040/Maze-Solver-shortest-path-finder [Accessed 12 Dec. 2017].

Antimodal.com. (2017). *astar*. [online] Available at: http://www.antimodal.com/astar/ [Accessed 12 Dec. 2017].

Carbonell, J.G., Michalski, R.S. and Mitchell, T.M., 1983. An overview of machine learning. In *Machine learning* (pp. 3-23). Springer Berlin Heidelberg.

Cheng, K.T. and Krishnakumar, A.S., 1993, July. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th international Design Automation Conference* (pp. 86-91). ACM.

Cole, N., Louis, S.J. and Miles, C., 2004, June. Using a genetic algorithm to tune first-person shooter bots. In *Evolutionary Computation, 2004. CEC2004. Congress on* (Vol. 1, pp. 139-145). IEEE.

Counter Strike. (2017). [image] Available at: https://upload.wikimedia.org/wikipedia/en/archive/0/02/20171204060818%21Counter-Strike_screenshot.png [Accessed 12 Dec. 2017].

Cui, X. and Shi, H., 2011. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, *11*(1), pp.125-130.

Daylamani-Zad, D (2017). Artificial Intelligence in Games. Lecture, [Online], Slide 25, 44 and 51.

Dynamic Learning. (2017). *Chapter 13 – Dijkstra's shortest path algorithm*. [online] Available at: https://resources.dynamic-learning.co.uk/Titles/AQAALCompSciTL_9781471859212/31ef8a0d-c673-45cd-90f0-0be24cfe4d64/Resources/AQA_A_Comp_01680.pdf [Accessed Dec. 2017].

Edelweiss, N., Hubler, P., Moro, M. and Demartini, G. (n.d.). A temporal database management system implemented on top of a conventional database. *Proceedings 20th International Conference of the Chilean Computer Science Society*, p.pg. 59.

Georgeson, J. and Child, C. (2017). *NPCS AS PEOPLE, TOO: THE EXTREME AI PERSONALITY ENGINE*. Ph.D. City University London.

Goldberg, D. (2012) *Genetic algorithms in search, optimization, and machine learning*, Boston [u.a.], Addison-Wesley, pp. pg. 1-3.

Havard University (2017). *AI in Video Games: Toward a More Intelligent Game*. [image].

Jensen, C. and Snodgrass, R. (1999). Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), pp.36-44.

Kumar, A., Tsotras, V. and Faloutsos, C. (1998). Designing access methods for bitemporal databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(1), pp.1-20.

Lara-Cabrera, R., Nogueira-Collazo, M., Cotta, C. and Fernández-Leiva, A.J., 2015. Game artificial intelligence: Challenges for the scientific community.

Lou, H. (2017). *AI in Video Games: Toward a More Intelligent Game - Science in the News*. [online] Science in the News. Available at: http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/ [Accessed 11 Dec. 2017].

Ma, J (2017). Temporal Representation and Reasoning
in Artificial Intelligence. Lecture, slide 6, 43, 46, 49, 52, 53 and 56.

Maulik, U. and Bandyopadhyay, S. (2000) Genetic algorithm-based clustering technique, *Pattern Recognition*, 33(9), pp. 1455-1465.

Millican, P., Clark, A., (eds.) (1996). Machines and thought-The legacy of Alan Turing. Oxford: Clarendon Press

Ntu (2017). *MinMax*. [image] Available at: https://www.ntu.edu.sg/home/ehchua/programming/java/images/GameTTT_minimax.png [Accessed 13 Dec. 2017].

Parker, L. (2010). *The Future of AI in Games*. [online] GameSpot. Available at: https://www.gamespot.com/articles/the-future-of-ai-in-games/1100-6283722/ [Accessed 11 Dec. 2017].

Policonomics (2017). *Prisoner's Dilemma*. [image] Available at: http://policonomics.com/wp-content/uploads/2016/02/Prisoners-dilemma-Nash-and-Pareto-equilibria.jpg [Accessed 13 Dec. 2017].

Snodgrass, R., 1987. The temporal query language TQuel. *ACM Transactions on Database Systems (TODS)*, *12*(2), pp.247-298.

Snodgrass, R.T., 1992. Temporal databases. In *Theories and methods of spatio-temporal reasoning in geographic space*(pp. 22-64). Springer, Berlin, Heidelberg.

TimeConsult (2005). *Temporal Databases*. [image] Available at: http://www.timeconsult.com/TemporalData/TemporalDB.html [Accessed 15 Dec. 2017].

Wachowicz, M. (2003). *Object-oriented design for temporal GIS*. London: Taylor & Francis, p.pg. 40.

Watson, I., Azhar, D., Chuyang, Y., Pan, W. and Chen, G., 2008. Optimization in Strategy Games: Using Genetic Algorithms to Optimize City Development in FreeCiv. *Interim Report. Damir Azhar*.

Wexler, J. (2017). Artificial Intelligence in Games. [online] pp.pg. 4-5. Available at: https://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf [Accessed 11 Dec. 2017].

Youtube (2017). *Call of Duty WW2: Dumb Yet HILARIOUS Glitches*. [Video].