An Introduction to Artificial Intelligence and Real Time Strategy Games

Big Data and Machine Learning have become big topics in computer science. Big Data can be traced back to concepts in database management and today's tracks in machine learning can be traced back to the coining of the term Artificial Intelligence.

Big data is a buzzword used to describe a massive volume of both structured and unstructured data that is so large that it's difficult to process using traditional database and software techniques. Today, in most enterprise scenarios the data is becoming too big or it moves too fast or it exceeds current processing capacity. Technologies surrounding Big Data has the potential to help companies improve operations and make faster, more intelligent decisions in the future. Big data is being generated by everything around us at all times. Every digital process and social media exchange produces it. Systems, sensors and mobile devices transmit it. Big data is arriving from multiple sources at an alarming velocity, volume and variety. To extract meaningful value from big data, you need optimal processing power, analytics capabilities and skills.

Research in Big Data asks many questions. Some are the following:

- How can computation be employed to help process huge amounts of data and information to gain insight and knowledge?
- How can computation be employed to facilitate exploration and discovery when working with large amounts of structured and unstructured data?
- What opportunities do large data sets provide for solving problems and creating knowledge?
- How are algorithms implemented and executed on computers and computational devices?
- What kinds of problems are easy, what kinds are difficult, and what kinds are impossible to solve algorithmically?

In implementations of Big Data, computers are used in an iterative and interactive ways when processing digital information to gain insight and knowledge. Processing these data sets require multiple processor and multiple storage repositories working in parallel. The outcome of this is that large data sets can be processed to provide opportunities for identifying trends, making connections in data, and solving problems.

In order to begin understanding AI, machine learning and how it relates to gaming, one must understand the foundation on how any program works on a computer. A computer program is an algorithm. Linear Sequencing, selection sequences, and repetition sequences are the building blocks of algorithms. Linear Sequencing is the application of each step of an algorithm in the order in which the statements are given. Languages for algorithms include natural language, pseudo code, and visual diagrams and textual programming languages. In this lesson students will opportunities to write algorithms to solve basic high level operations in game development.

In today's society the participants in the society have to understand facets of Artificial Intelligence (AI) to contribute to everyday activities in life. There are many modern technologies that implements AI. A few examples are **Graphing Technology** which powers delivery services such as digital phone service, data services such as the internet and GPS services. We see during **Machine Learning** activities in Web Site Interaction. An example of machine learning during a human interaction with the web site is when you visit websites where they can dynamically adapt to the profile of specific people. When you purchase items from an e-commerce site and the site specifically knows your buying patterns and suggests additional products that you may be interested in. Finally there is everyone favorite computer games. In many computers games, human players interact with computer players. The computers player can be set to have varying levels of difficulty. This can be set by the human player or progressive levels of difficulty can be conquered as the human player progresses through the game.

This document provides an introductory lesson on artificial intelligence and reviews material learned in a research project examining a Real Time Strategy Game titled MicroRTS. One of the branches of the research was to discover how MicroRTS can be used to inject computer science research into teaching computer science topics into lessons in the class classroom. This research activity tries to ascertain can MicroRTS, a Real Time Strategy Game, be used and integrated as an engagement tool into a lesson that teaches fundamental AI concepts. It will also show how big data and AI machine learning correlate

The focus of this lesson is to cover brief elementary overview of artificial intelligence, and a few fundamental artificial intelligence techniques for computer games. This document includes an Introduction to AI, an introduction to Real Time Strategy Games (RTS) gaming and its basic behavior. Additionally, it includes examples of game AI applications. Finally it will review a few basic AI game techniques in the area of RTS gaming.

There are many algorithms and research activity in artificial intelligence, machine learning, and game development. The scope of this lesson is an introduction to AI and RTS as it related to MicroRTS. In particular it examine hardcoding intelligence into the application using selection and repetition sequences. We will briefly discuss techniques that can be used to let the AI autonomously take decisions, however these techniques should be learned in detail at higher level courses that specifically focus on game development.

The first use of the term Artificial Intelligence

The first use of the term Artificial Intelligence was used by John McCarthy. John McCarthy was a computer scientist and cognitive scientist. Dr. John McCarthy was one of the founders of the discipline of artificial intelligence. He coined the term "artificial intelligence" (AI), he developed the Lisp programming language, significantly influenced the design of the ALGOL programming language, popularized the concept and use timesharing, and was very influential in the early development of AI.

“We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.” Computer Scientist - John McCarthy, August 31, 1955

What is Artificial Intelligence?

- An area of computer science that deals with giving machines the ability to seem like they have human intelligence.
- The power of a machine to copy intelligent human behavior

Artificial intelligence (AI) is the human-like intelligence exhibited by machines or software. It is also an academic field of study. Major AI researchers and textbooks define the field as "the study and design of intelligent agents", where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1955, defines it as "the science and engineering of making intelligent machines".

http://www-formal.stanford.edu/jmc/whatisai/

Examples of Artificial Intelligence in Action

Deep Blue 1997 Computer defeats Chess World Champion See more detail by clicking on the video link

Chess Match Video

Watson 2011 Computer defeats Jeopardy Champions See more detail by clicking on the video link

Jeopardy Video

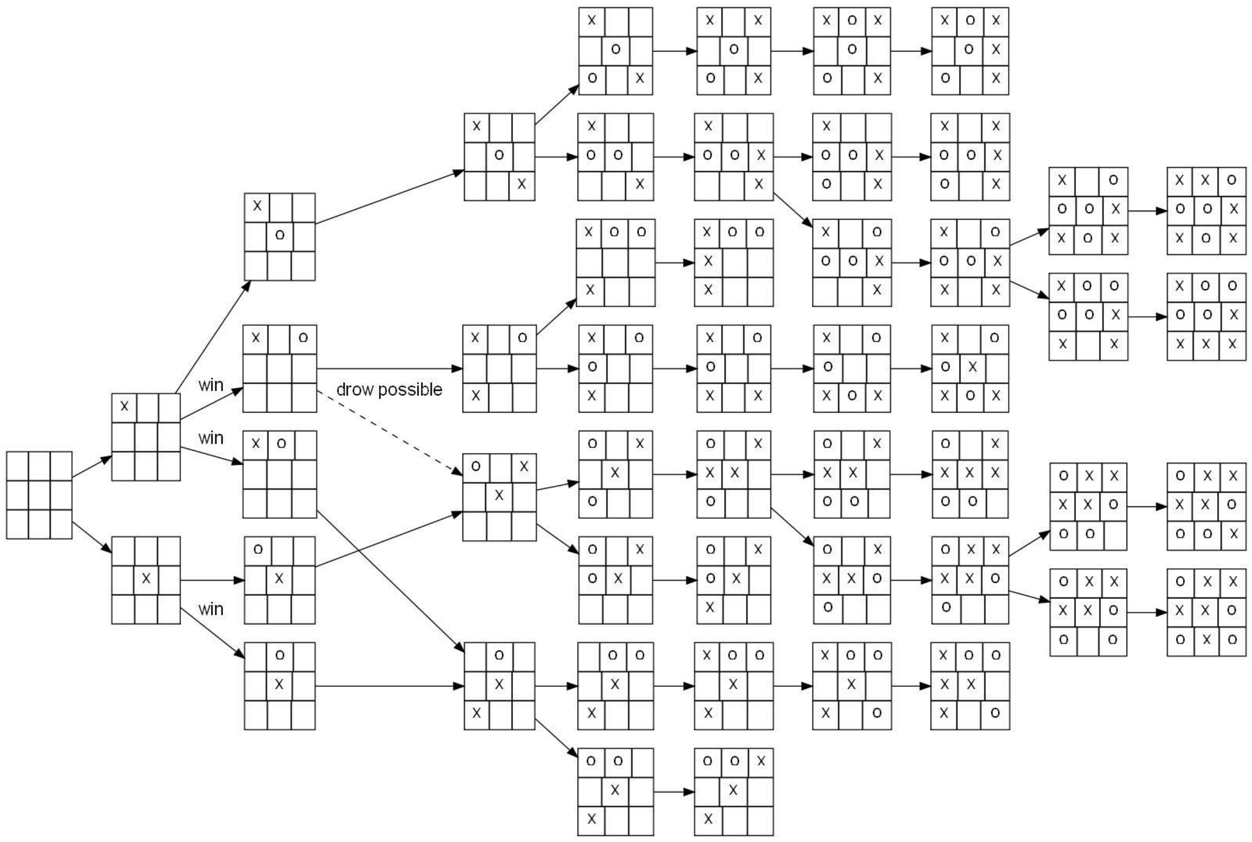
In Game theory there are perfect information games and imperfect information games. In a perfect information game, when making a decision, a player is aware of all previous moves as well as the current state of the game. An example of this is the game of chess. In an imperfect information game, Players are missing data about their competitor's "state", strategy, preference, an example of this is a poker game.

Are computers smarter than humans? Think about a TIC TAC TOE game. Can you make an algorithm make your computer never loose when going second?

Examine the link below to help you answer that question. It points to a website that opens a document tilted "The Intelligent Piece of Paper"

The Intelligent Piece of Paper Link

This figure shows how the computer would react to just two different moves the user can make and then how it evaluates all its possible moves. This is only a subset of what the computer has to go through since there are 7 more options the user would initially have. It should demonstrate how intense the decision making process is for the computer and the thought programmed into each scenario for the computer. Computers have to handle a large subsets of data coming in and then process that data quickly. Computers have to make complex decisions using IF statements to choice a path to take based on conditions in the game. This is also referred to as the state of the game. How does the computer know which move to make in order to increase its chances to win (we will get to that later).

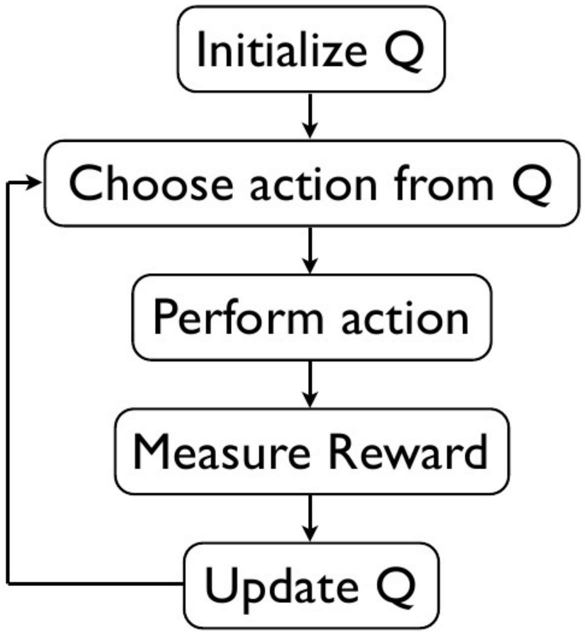


How have systems/games we have discussed worked so far? How did Deep Blue defeat the human chess champion? How did Watson defeat the other human Jeopardy player? How does a computer TIC TAC TOE game work?

One of the main focusses of this lesson is Real Time Strategy (RTS) Games. What about games that are played in real time, where both players are acting simultaneously? As a programmer, how can you predict these changes and create algorithms to respond? Can we create an AI Agent that can improve itself based on successes and failures? These are the types of questions that one would ask if they were a computer scientist working in AI.

How would an AI Agent improve itself? Let's examine an abstract algorithm for accomplishing learning or improvement. The AI Agent can randomly select an action the first time a state is reached in the game. Based on the impact/result of that action, whether the computer wins or loses, the computer increases or decreases the likelihood of that action being chosen next time that given state is reached. If result was negative, the computer loses after taking the action, next time that state is reached the computer will choose a different action and determine that action's effectiveness. This process continues for all different states and actions. The computer can track the results over time in a table. When the computer plays again, it can reacted based on past experience. Isn't this one way we learn as humans. Once enough data has been gathered AI Agent will know optimal moves to play in response to given states. The best move will change as new data comes in, until a significant confidence level has been created that given a state one can identify the best move. This is not going to happen until a large enough data set has been collected and tested. You now have a machine that after playing millions of times has learned optimal behaviors just like a human who plays enough to learn how to beat a game. The computer is LEARNING! The problem is that the computer has to play a large number of sets to learn enough to become competitive, sometimes millions of games. The learning curve is mitigated by having the computer play against itself with its opponent using a different plan of attack.

Here is that abstract model diagramed below. At the beginning of the game, The AI agent initializes by loading the historical data into a lookup table. The table contains a mapping of successes and failures. The AI agent chooses an action from the table based on past experiences being in that position. The agent takes the action and then measures the impact of the action. The agent then updates the table based on the success or failure of the action. This processes is repeated until the game is complete.



Below is a link to a document that contains directions to an AI Learning Game. The document is titled the Sweet Computer that Learned. The document is also packaged at the GITHUB web link for this research project. You may alternately download the document form the GITHUB site

<u>Sweet Computer that Learned</u>

In this lesson, you are going to have some fun. Part of this lesson is to actually play a RTS game. The RTS game was developed by Santiago Ontanon, a computer science professor at Drexel University. One major area of research of his is Real-Time Strategy Game designed for AI research. During the face to face lecture students participating will get a demonstration by the instructor to see how the game works. After completing the game students will be asked to complete corresponding follow up questionnaire. You should study the accompanying Instruction manual on how to play the game. If you thinks that you are up to it, you can down load the game and configure a Interactive Development Environment (IDE) to run the game from home. Download and configuration instructions can be found at URL Link below.

https://github.com/can39/TestProject

In the genre of AI gaming, there are three types of artificial intelligence in gaming. The three areas are;

1. **Inside** the game:
   - Character control
   - Director (drama management)
2. **During** game development:
   - Help in behavior/content design
3. **After** game deployment:
   - Analysis of game data

The area of AI gaming algorithms, this lesson will focus only on fundamental aspects of character control in real time strategy games through hard coding using selection and repetition sequence.

A few Basic AI Techniques that will be covered in this lesson are;

Scripting:	Finite State Machines
Path-finding:	Path finding for navigating in complex maps. E.G Shortest Path, we will review only shortest path
RTS games:	movement behaviors, attack behaviors

Algorithm Development

One of the goals of this lesson is to understand or reinforce basic concepts basic algorithmic development and basic algorithmic development as it pertains to RTS game development. We will use game agent movement behavior and game agent attack behavior as artifacts to explain algorithm development and some RTS behavior. We will examine some situations in the MicroRTS game to explain fundamental movement algorithms in real time strategy games. We will also try to examine situations where may work or not work. One of the goals of the exercises in this lesson is try to get students to describe why or why not an algorithm will work in either case. In addition to having student design their own algorithm, student's will be presented with basic fundamental AI algorithms and ask what could be done to improve on them. As an exercise, participants will have to describe a basic algorithm for an AI agent in a real RTS game. The target game for this exercise is MicorRTS a fundamental operational framework of a real time strategy game.

Object Oriented Programming Concepts

MicroRTS is written in the Java Programming Language. Java is an example of a high level Object Oriented Programming language. There are many opportunities in the code of MicroRTS to illustrate the application of using OOP concepts in coding. The code uses of inheritance, composition and polymorphism. During this lesson the participants will examine a software professional application, and see illustrations of principles in object oriented programming in use. To accomplish this, students will be shown code snippets of MicroRTS where inheritance, composition and polymorphism is being used. During the face to face class the participant are given an explanation of what is occurring in the code.

For Example: *Place Code here*

Real Time Strategy Games

A principal goal of real time AI video game development is the creation of intelligent agents. In war games the intelligent agent may serve as enemies or allies. The role intelligent agents vary from game to game. A requirement of the intelligent agent it that it must react to the environment in the playing. In MicroRTS the environment is composed of the (left, right, top, bottom) barriers, military units, buildings and workers in the game. In MicroRTS, the actual intelligent agents are represented as workers and military attack units. if a worker or military attack unit is patrolling and an enemy unit comes near, the agent is programmed to stop patrolling and attack the enemy unit. This is an example of an intelligent agent reacting to the environment. This is an important part of the behavior of intelligent agents in real time strategy games. We can use this opportunity to explain how a searching algorithm such as spiral search works:

Let's examine a basic architecture and design of a RTS game. Let us examine the components of this type of architecture. RTS games have a level of perception for the user. There is a strategy level and there is a command level. The perception level provides a survey of the playing area, the unit deployment on the playing area and feedback such as the scoreboard. RTS games have a common architecture that includes having an economy. The strategy of the RTS games are is to combine economy, logistics and combat to win the game. Later in this document student's will be provided with the basic architecture of a RTS game and examine how MicroRTS meet the minimum requirements of the architecture off an RTS game.

Additionally, RTS games have aspects of artificial intelligence embedded into them. Typical AI features are path finding capabilities, unit roaming, and building placement capabilities. Many of these types of AI behaviors can be achieved using rules expressed as selection sequences and repetition sequences in program code. The results of passing requests to these rules are sent to a logistic module that actually performs the operation. There is a logical unit named an arbiter module that enforces the rules of the game such as ensuring that actions are only executed when there are enough resources are collected to create the unit

Another characteristic of a real time strategy game is the capability of maintaining Resistance capabilities, attack power and remaining health levels. These features are incorporated into units in a RTS game. The user is provided with visual indicators of the state of the features. These features are monitored and directly feed into interactive environment of the RTS game

In RTS games weapons can be fired over ranges, weapons can fire over ranges.

Place explanation here: *Spiral Search*

Intelligent Agent Movement

Intelligent agent movement, in RTS game, an agent needs to get to a particular destination. The agent should be able to figure out a path to get to that particular destination. The agent uses a path finding algorithm to return the best path to travel to get to its appointed destination. There are a variety of algorithms that a game developer may choose from. This document points to supplemental resources where students can obtain additional information on algorithms to find paths from one point to another. In our discussions on it in this lesson we will view it as a function that magically returns the path to an agent. Algorithms such as shortest path algorithm are beyond the scope of this lesson. In this lesson, students are shown basic operation during game play and will get presented the opportunity of describing an abstract algorithm that can perform the movement in the game using starting point and ending point as parameters to their algorithms.

Artificial Stupidity

Another key aspect of giving intelligence to intelligent agents in a game is that the agents must be provided with the appropriate level of intelligence. In traditional AI, the ultimate goal is optimizing and maximizing the intelligence of the AI agent. In game development the developer strives to obtain artificial stupidity. The goal in a video game is to allow the player to win, while providing challenging and interesting experience. A stupid AI character that the player can beat all of the time or a player that is inhumanly smart will not hold the interest of the human player.

There is a fundamental difference between gaming AI and Traditional AI. In traditional AI obtaining optimal efficiency in simulating human intelligence is the goal. In contrast, in gaming AI the goal many times is to add levels of stupidity to the AI agent. It is not much fun playing a game where the computer always wins. Additionally, this permits one to have different levels of difficulty. In the lesson, we will only briefly talk about the difference between traditional AI and Gaming AI. In Gaming AI the goal is artificial stupidity. There are many algorithms describing higher level details of traditional AI and Gaming AI, the scope of these topics are far beyond this basic introduction lesson to AI to RTS games

Data-driven design

Data-driven design is a programming technique where, outside of the program, data exist that can be modified to change and tune the behavior of the game and the AI. One can look at this in a simple form as allowing a user to select a level or playing environment. The data that sets the AI agent to particular level of intelligence is stored in an external storage space. The software will use decision of the human user to select (movement and attack algorithms) imbedded in the code or read data from a file to provide a level of intelligence and the construct the playing environment. Participants in the lesson will be shown example code of programs reading basic a text file and reacting to the contents of the data in the file. Basic selection sequence are used to make decisions that will display varying information based on the data.

Decision Theory

Decision Theory - When automating AI when the unit is given a situation, it must decide which action to perform depending on the desirability of its immediate outcome. The real goal of Gaming AI is to get the automated player to autonomously make decisions. To make decisions on its own, and can generate strategies, not foreseen by game designers. It must decide which action to perform depending on the desirability of its immediate outcome. There is typically a mathematical functions that calculates this. The details of this level of autonomous decision making is outside of the scope of this introductory lesson. These skills and knowledge can be obtained in a series of courses that focus on game development.

Decision making in a program execution is a choice of possible actions to execute. It is up to the programmer to set up a decision process to determine the best possible action in the RTS game. Typically the type of data that is collected is player data or keeping a history of explored terrain. The use of arrays in programming is illustrated to provide a framework on how a player's movement can be recorded and retraced for later use.

Real Time Strategy Game Behavior

In object oriented programming in Java we like to examine an object's behavior. A behavior of an object can looked at being a sequence of actions or tasks that the object can perform. One of the reason why advances in artificial intelligence has lagged is the limitations of the computer hardware that must support it. Computer hardware is becoming more powerful and capable to handle the huge resource demand of AI algorithms. It is now possible to write complex game AI behaviors using a variety of programming techniques that were not practical to do several years ago. This includes some theories from AI research that were theoretically sound but were never actually tested. The appendix will point to resources on some of these topics. However In this lesson, we will look at creating algorithms that allows for the creation of building from resources and algorithms that controls the sequencing of an ag An AI system that creates behaviors as a sequence of actions must be able to generate different, appropriate sequences based on the circumstances in the game agent's behaviors

Controlling Groups of Agents

Another difficulty in designing and implementing RTS games is having to manage and control many units on the playing area simultaneously.   In an advance lesson, students are asked to participate in a brief discussion on deploying units in squads. This is beyond the scope of the basic introduction lesson on AI and Real Time Strategy Games.

Another behavior of a typical real time strategy game is the ability to guide units in roaming activities. This type of behavior is useful when roaming a unit, randomly selecting an automated location finder for the construction of buildings or a base. When implementing a roaming algorithm, a spiral search algorithm can be used.

Another behavior of RTS games is while a unit is roaming, actions can be taken when a roaming unit is close to a resource or close to another enemy unit.

The MicroRTS Design

In the MicroRTS games we see that it uses the divide and conquer strategy of modular development. There are different modules to handle different sub problems; such as a module for path finding, and a module for construction of units. The code in MicroRTS provides an opportunity to illustrate and reinforce the significance and importance of modular development. In this less students are guided through an illustration of modular development and walkthrough of the MicroRTS code identifying the use of modular development.

Let's take a look under the hood of MicroRTS . In MicroRTS the worker units and the resource buildings represent the economic factor of the games. The role of the workers is to collect resources to build things. The workers are responsible for building bases and barracks. The worker must be guided by the player to collect resources and to build a building at a particular location on the board. A base building is used to store gathered resources that the worker unit collects. Once a base building has resources, it can be used to create additional workers from the gathered resources. That is one of the roles of the base building, to create additional workers for the game. A barracks building use gathered resources to create military units. In order to combat the AI agent representing the opposition player in game, the human player must create military units from barracks building. RTS games have a common architecture that includes having an economy that is used to obtain resources to construct artifacts that are useful to the maintain the existence of the player by providing means to construct building to build things, resources to use to build things, workers to use to build obtain resources to build things and military units to defend and attack.

MicroRTS represents the basic architecture and design of a RTS game. Let us examine the components of this type of architecture. RTS games have a level of perception for the user. There is a strategy level and there is a command level. The perception level provides a survey of the playing area and the unit deployment on the playing area. In MicroRTS this is the playing grid and the feedback of the scoreboard. The strategy of the RTS games are is to combine economy, logistics and combat to win the game. Students are provided with the basic architecture of a RTS game and examine how MicroRTS meets the minimum requirements of the architecture off an RTS game.

As mentioned earlier, RTS games have aspects of artificial intelligence embedded into them. In this lesson students will examine how MicroRTS establishes several layers of AI. Additionally, students will examine code and toggle from one AI strategy to another. Typical AI features are path finding capabilities, unit roaming, and building placement capabilities. Many of these types of AI behaviors can be achieved using rules expressed as selection sequences and repetition sequences in program code. The results of passing requests to these rules are sent to a logistic module that actually performs the operation. There is a logical unit names an arbiter module that ensures that actions are only executed when there are enough resources are collected to create the unit

As mentioned, another behavior of a typical real time strategy game is the ability to guide units in roaming activities. This type of behavior is useful when roaming a unit, randomly selecting an automated location finder for the construction of buildings or a base. When implementing a roaming algorithm, a spiral search algorithm is typically used. This is the type of algorithm that MicorRTS uses.

While a unit is roaming, actions can be taken when a roaming unit is close to a resource or close to another enemy unit. MicroRTS displays this type of behavior when deploying units and placing bases and barracks. The class will discuss this and review this time of behavior in the actions of the units in MicroRTS

Setting, updating and displaying Resistance capabilities, attack power and remaining health are features that are incorporated into units in a RTS game. The user is provided with visual indicators of the state of the features. These features are monitored and directly feed into interactive environment of the RTS game. In MicroRTS resistance capabilities, attack power and remaining health are factors built into units in the application. Each unit has resistance capabilities, and attack power. Additionally they have remaining health indicators if they can resist more than 1 hit form an enemy unit. Refer to the class description for the basic properties required for a movable unit in a RTS game.

Basic Algorithms in a RTS Game

An important aspect of AI behaviors is that the actions in the sequence vary depending on the circumstances in the game. " This can lead to enormous overhead cost in the performance of the application. This is something to take note of, however studying solutions to this is for a more advance lesson.

In a RTS game let us look at possible AI behaviors that one would need to implement; patrolling an area and guarding against an enemy. These behaviors would consist of the following actions: Move, Scan, Acquire Target, Approach Target, Fire, and Detect Damage. This section describes the methods or program modules that are needed for basic military operations in the game. In RTS games weapons can be fired over ranges, weapons can fire over ranges. This can be seen in the Range Unit in MicroRTS. In RTS games, opposing units are destroyed after a fix number of hits. There are also indicators determining the damaged that units have taken.

Let's examine an API for a RTS game;

A MoveTo method: this method moves an agent from one position in the game to a destination location.

An Attack method: this method locks on to an enemy unit in range and attacks the unit

An Approach method: Moves an agent to a location suitable for firing once an enemy agent has been detected.

A Aims method: Points a weapon at a target and verifies target is in range.

A Fire method: Fires the weapon at a target once aim has verified target is in range.

A CollectResources method: Automatically collects and delivers resources.

An Idle method: places unit in an idle state to wait for commands.

A Guard method: Holds a position and attacks any enemies that approach.

An AI system that creates behaviors as a sequence of actions must be able to generate different, appropriate sequences based on the circumstances in the game.

Let's examine the required tasks to perform a move operation: Once a unit has been designated to move to a new location the path that will follow getting to the location has to be processed. A Request to a method for a path to follow has to be initiated. The unit will then wait for a response to the request. The unit receives data on the Path to follow to get to its destination, The Move method is called to move the Agent, There are two goals the agent will an intermediate goal and a final goal. Check Intermediate Goal, and Check Final Goal

Transitions: The transitions are potential changes from one state to another, but only one can happen at a time. For example, in the MoveTo state, no transitions will be generated until the agent reaches the goal position.

Finite State Machines

Finite State Machines (FSM) – the first commercially successful example of a computer game that used AI as a FSM was PAC MAN.

Let's examine the Finite State Machine. A FSM can be represented as a sequence of binary choices in a program algorithm. In a basic RTS game some of the most common commands are attack, build, explore, and harvest resources. The FSM can become unmanageable for complex tasks. Complex task handling can be explored in a more advanced lesson of AI and RTS game development. In fact the FSM structure itself may be slightly beyond this basic lesson on AI and RTS gaming. However, let's try to get an overview of it anyhow.

There are two ways for transitions to be generated for FSMs: internal and external. Internal transitions are generated by the states inside the FSM and often occur when a state completes an operation.

Here is an example of an internal transition, the active state may generate a transition to another state. For example, a move state may transition into an idle state. Another example is when performing a military operation such as an attack, the aim state will transition into the state to fire the weapon.

Let’s examine an example of an external transition. The external transitions can be generated externally or outside the FSM. For instance, in an RTS game a player may click to command an agent to move to a new location, this would be considered an external stimulus. An additional example, the player may need to destroy an enemy target. Here is the sequence of activities. The player selects their military unit to conduct the attack and then clicks on the enemy target. Let’s examine what this triggers. Inside the game, the click generates a transition to the Approach state in the military units AI component. The unit then moves to the chosen target, aims, transitions to fire, and begins firing.

Let’s take a look at activity from a higher level of abstraction. Activity in games are contained in cycles. There are many activities that can be placed into a game cycle. Here is an example of a loop sequence to process the tasks in a cycle of the game. When processing a cycle, the input system is updated to check buttons pressed by the player, this is referred to as an interrupt; the movement system is updated to move the game objects for that cycle; the AI system is updated to calculate and perform the active AI behaviors for the units in play; it is beneficial to make the execution as consistent as possible. It can make a difference whether the processing occurs before the Move Units system has updated an object’s position, or afterwards.

Now let us look at handling more sophisticated and lengthy sequences of actions using a FSM. We can examine it from the context of object oriented design. There is an object that represents the State Machine. There is another object that represents the state of the game at any one cycle in game play. The State Machine class consisted of a list of states and a pointer to the currently active state. Each state is an instantiated instance of an object of class named State. The State Machine class is a controlling class that keeps track of the current active state and handles any transitions between states. One can view this as motion picture film, a series of still images presented to the viewer rapidly, giving the appearance of smooth motion. In the physical world, movies show 24 images per second, and some video displays can show up to 60 images per second. Each image is called a frame. A video game must generate at least 30 frames each second to produce the appearance of smooth movement on the display. The state class can be viewed as a frame is the sequence of states representing a fluid flow of action in the video game. Similar to a movie or video display, at every state, after processing the game

AI, the state machine transitions to the next pending state object in the collection state objects representing transitions to the next sequence in the game.

Now let’s look at it from the view point of an event driven program. When the player clicks on the mouse to direct the agent to move to a particular location, a user event is invoked, queued and processed. This results in a call to the StateMachine’s TransitionTo(MoveTo) method. The StateMachine variable will refer to an object of class State. Another of the methods for the StateMachine class is the TransitionTo() method. This method is used to direct the StateMachine object to make a particular state in the array of states, the active state In a method call, the parameter (MoveTo), is the pointer to the state that will be activated. Next in sequence the StateMachine’s Update () method is invoked, the update method will process the sequence of actions associated with the next state in the list.

Let’s describe the behavior and life cycle of a state object. It will have an onExit () and an onEnter () method. When state object is invoked a call is made to the on Enter () method when a state object has finished, its Onexit () method is invoked. There are different states such as an “idle” state, and a “move to” state. Typical operation of the move to state can be storing destination position on the board, calling the pathfinder method to return a path to follow to the destination position. The pathfinder method returns an array object with a list of intermediate positions that may be followed to the destination position. This list may be modified after each state change. In the case of an open field move, the path would be two points—the agent’s current position, and the destination position. A list of intermediate sequential moves re returned. In the event there are structures or units, such as bases, barracks workers or military units on the playing grid, the array containing the path would contain more points to go around these obstacles.

The StateMachine object calls the update method of the MoveTo state once when processing each frame. This allows the MoveTo state to perform the appropriate processing to execute the move to behavior. The code in the update method of the move to state looks at the current position, checks the next intermediate destination point in the path that it is following in the movement array, and updates the unit current position to the intermediate position, on its way to the final destination. Once the agent reaches the intermediate position point, it starts moving toward the next intermediate position point. When the unit reaches the end of the path, or the final destination, it generates a transition back to the Idle state, since the movement is now be complete. The transition is generated by calling the

StateMachine’s Transition method TransitionTo (Idle), with Idle as the parameter. This results in the Idle state becoming the active state in the collection of possible states.

Game Movements and agent AI can be handled using complex decision sequences.

Example of Complex Decision Algorithms

1. If we have not seen the enemy then build ground units and we have more units than our enemy, then attack.
2. If enemy units have been detected and the enemy has more units than agent player then build more units

Decision Trees are easy to implement:

In the context of assignments, decision trees are used as “paper and pencil” technique, to think about the problem, and ta person can use nested if-then-else statements to work out a solution to the problem

Attack Mode Behavior- let’s examine a possible algorithm for conducting an attack on a target

Aim: The Aim state consists of pointing the weapon at the target. The Aim state completes when the target is within the parameters of the weapon. While aiming, the target agent may be moving away from the attacking agent and will move out of range or behind a base or barrack. In this case, the Aim state will transition back to its move state in order to move back into range.

Fire: Waits for notification from Aim that it is ok to fire.

Move: Moves to an adjacent square and determines any enemy units are in range

Approach: The code in the Approach state has to be able to handle movement and determine a destination in range of the target.

AI and Supervised Learning

Supervised learning is the machine learning task of deducing a conclusion from a function using labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair of data consisting of an input object (usually a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm examines the training data and produces an inferred function, which can be used for creating new examples not previously contained in the training data. This is how artificial agents can learn new options to take that have never been used. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to produce unseen situations.

In order to solve a given problem of supervised learning, one has to perform the following steps:

1. Determine the structure of the training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In the case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.

2. Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.

4. Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.

5. Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a *validation* set) of the training set, or via cross-validation.

6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

Learning from Observation (LFO)

In learning from observation, the learning agent A first observes one or several actors performing task T in the environment E, and records their behavior in the form of traces. Then, those traces are used by A to learn how to perform T via a learning algorithm.

Task, T: a task to be learned, which can be either achieving a condition (such as building a tower, or defeating an enemy), maintaining some process over time (such as keeping a car on the road), or maximizing some value. In general, T can be represented as a reward function, which has to be maximized.

Environment, E: simulated or real, where the task is performed.

Actor (or trainer or expert), C: who performs the task T in the environment (there can in principle be more than one actor).

Learning Agent, A: whose goal is to learn how to achieve T in the environment E by observing C. Some works in literature focus on multi-agent learning from observation; in this lesson, we will focus on single-agent learning from observation.

The environment in which the learning agent observes the actor perform the task need not be the same one in which the agent later performs the task. For instance, in the context of a computer game, the learning agent might observe the actor play the game in a particular map, and then try to perform the actions in a different map. Moreover, most LFO work assumes that the agent does not have access to a description of T during learning, and thus, behavior must be learned purely by unobtrusive observation of the behavior of the expert.

Actor C might have non-observable internal state. For example, consider the Stratego game, a strategy game similar to Chess, but where the players do not see the types of the pieces of the opponent, only their locations. After certain movements, a player can temporally observe the type of one piece, and must remember this in order to exploit this information in the future. A person playing the game of Stratego might keep an internal record (in memory) of the type of known pieces that the opponent might have on the board – information not directly observable. Because the behavior will be influenced by this memory, it is necessary to somehow include it in the model. This is accomplished by recording this observed information in a data structure such as an array in the design of the game

Levels of AI Behavior

Level 1 – Strict Imitation:

Some behaviors do not require feedback from the environment, and thus require neither generalization nor memory. The learned behaviors are a strict function of time. For that reason, it does not matter whether the environment is observable or not. One can think of these behaviors as requiring open-loop control. Robots in factories are an instance of this type of learning, where the pieces are always in the exact same place each time.

Level 2 – Reactive Behavior:

Reactive behaviors correspond to input-to-action mappings, without requiring memory of past events. Generalization is desirable when learning these behaviors. Reactive behaviors are Markovian, because they can be represented as a mapping from the perceived state of the environment to the action, i.e., a policy. Learning how to play some video games such as pong and space invaders would fit this category.

Level 3 – Memory-based Behavior:

When the current state of the world is not enough to determine which actions to execute, previous states might have to be taken into account. These behaviors are not a situation-to-action mapping, but might have internal state. For example, learning how to play a game like Stratego (described above) is a task of Level 3. Thus, a learning agent has to take into account past events in order to properly understand why the actor executed certain actions

What Algorithms are useful for Levels 1 and 2??

While level 1 problems can be considered trivial, learning behaviors of levels 2 or 3 from observation constitute the most interesting cases. Traditionally, most work on LFO has focused on level 2 (or has simply ignored the distinction between level 2 and 3 completely).

It is important to distinguish level 2 from level 3, because the kind of algorithms required to learn those behaviors are inherently different. This distinction is typically not made in the LFO literature, and standard supervised learning algorithms or reinforcement learning algorithms have been and are used to learn tasks of level 3.