Alex Dulmovits

Luis Villegas

CSE 348

Real Time Strategy Games AI

RTS History

One of the most popular game genres (especially on PC) is Real Time Strategy (RTS). It is mostly recognized for its unique type of gameplay where participants position and maneuver units and structures to secure areas of the map and/or destroy their opponents' assets. Many if not most RTS games have a focus on economy, where the player must use their resources strategically to stand a chance in a battle.

There were a few early games that had some RTS characteristics, such as Utopia which had direct manipulation of units and resources, but it was turn-based. Legionnaire was another RTS precursor that was actually a real-time game, but it didn't have any resource or economy/ production concepts. The first game to have all the modern core concepts of RTS games was Westwood's *Dune II: The Building of a Dynasty* (1992), which served as the prototype for later popular real-time strategy games in the 90s such as Warcraft: Orcs & Humans (1994), Command and Conquer, Age of Empires (1997) and Starcraft (1998). Many games have evolved the RTS formula, such as Empire Earth, which introduced researching to advance epochs. Sins of a Solar empire introduces grand-scale stellar empire building to the RTS genre with huge-scaled battles in 3 dimensions. Other games such as Achron change the formula even further with new features such as time travel.

Architecting an RTS AI

To take a general look at RTS AI, we must look at components that make up an RTS game architecture which are consisted of "managers". The main managers in an RTS AI are Civilization, Build, Unit, Resource, Research, and Combat. All these managers have sub managers that help organize the tasks.

Civilization Manager

The Civilization manager is the highest-level manager responsible for development of the computer player's economy. It is also responsible for coordination between the build, unit, resource and research managers. It has the responsibility to manage spending limits, the computer player's expansion, upgrading and epoch advancement (in games such as Empire Earth)

Build Manager

The Build Manager is responsible for the placement of structures and towns within the game world. It receives requests from the unit manager for training buildings and requests from the civilization manager for support buildings. All site evaluation occurs within this manager. In RTS games, there is restriction with component placement, so the Build Manager it decides where buildings can and cannot be placed. The Build Manager must also take into account "area of effect" which is the area in which a building effects units, whether it be to give higher hit points to friendly units within the area, or to increase productivity within that area. Buildings with an area of effect are placed near resources, where most units are likely to congregate. Military Buildings are put nearby resources, but shouldn't interfere with the "area of effect"

buildings. A strategy is to build the buildings in a ring around the immediate area of some resources. Other Buildings include farms, airports, walls, docks, and towers. These require specialized placements, but it depends on the terrain analysis engine used in the game. The manager usually places towers in choke points and tries to avoid building near enemies.

Unit Manager

The Unit Manager is responsible for training units and tracks what units are training in various buildings. It monitors the computer player's population limits and prioritizes unit training requests. If a training building that is needed not currently available, the Unit Manager communicates with the Build Manager so that the Build Manager will put the request for a new building in a queue. This manager attempts to maintain a similar unit count to the human players in the game to balance the difficulty of game. It also tries not to add too many units by not allowing AI unit count to get too high, which lessens the CPU load of the game.

Resource Manager

The Resource Manager is responsible for tasking citizens to gather resources in response from both the unit and build managers. Since every unit and every building needs some amount of resources, each respective manager must contact the resource manager. The Resource Manager also balances gathering duties and governs the expansion of the computer player to newly explored resource sites.

Research Manager

The Research Manager is responsible for the directed research of the computer player. Technologies are examined and selected based on their usefulness and cost. In games like Empire Earth, the Research Manager will also be in charge of the epoch advancement that results in new technologies available. This is important because depending on what units can be

queued, the strategy for the AI may change.

Combat Manager

The Combat Manager is responsible for directing military units on the battlefield and requesting units to be trained via the Unit Manager. The Combat Manager deploys units in whatever offensive or defensive position is most beneficial through its offensive and defensive sub managers. This manager keeps track of units via a personnel sub manager and communicates with the Unit Manager. The Combat Manager periodically updates the status, movement and actions of various attack groups.

Depending on what programming language is used, the combat manager communicates with the other managers in different ways. In C++, with the managers implemented as classes, public methods can be used as a means to communicate between all of them. All the game components will be communicating with each other at all times, so it has to be a seamless process. The problem comes when changing a manager after they have all been linked. Small changes have profound effects on AI and it becomes unpredictable. Tracking down chains of cause and effect becomes the focus of late development testing.

Difficulty Levels

Games are played by players with a broad range of experience, so it is a challenge for developers to tailor games to fit all levels of competence for the players. Developers need to find a way to create AI that can learn from the player as it goes along, but this is a hard and memory-consuming task, so they settle for pre-set difficulty levels. It's important to avoid just adding more computer players for more experienced players, since this is just more work for player

and for the CPU. The goal should be to develop an AI that is good enough to give challenge to an expert player in a 1-to-1 match, then tone down AI for intermediate and beginners mostly because it is more difficult to start with an easy AI and increase its intelligence Many times expert players are hired to help programmers do some balancing with the AI.

Developers need to choose what modifications to make to differentiate difficulty levels and then narrow them down to a few, usually in a brainstorming session. The more variables you have, the harder it will be to test changes, and testing difficulty level changes is very time-consuming. Many developers consider cheating, since its the easiest way to deal with the difficulty problem. The biggest challenge in programming AI and difficulty is dealing with random maps. The programmers cannot hard code AI if the maps are random, so they have to focus on terrain analysis, which will be explained later on. Because of all these challenges, RTS games are among the hardest to design AI for.

Prioritizing in a Goal-Based RTS AI

Goal-based AI differs from the usual AI systems, such as finite state machines and scripted AI since it isn't at all strictly controlled. Instead, goal-based AI lays out a structure of what AI should do in a given situation. Basing the AI off of goals also allows for behavior to become emergent and make it seem more believable. The way the goal system works is goals are attached to almost every possible action in the game and then are prioritized in a list where the goals with the highest priority are carried out first. This allows for a flexible and extensible AI that can pretty much "play on its own". Two RTS titles that use this system of goal-based AI are Kohan 2: Kings of War and Axis & Allies.

The goal engine architecture is responsible for controlling the actions of the computer AI (the team that the human player is versing). This architecture is broken down into basic elements. The first element is the actors. They are the units that are controlled by both the human and AI and have an affect on the game. Actors have many types, two of which are buildings and units. Buildings are immobile actors that can be either defensive structures or economic buildings. Defensive structures are buildings that can attack things and provide the player's town with protection from invasions. They can consist of buildings such as guard towers or barracks. Economic building provide a non military advantage , such as increasing the technology of a civilization or creating a new unit. Units are actors that can move. They are split into military units and builders. Military units can attack other units and builders are responsible for building and repairing buildings around the town. Each unit has a combat value (CV) which measures the unit's effectiveness in a fight. In some games the CV is straightforward for example a unit with CV of 50 can take out two units with a CV of 20. In other games, there is chance factored into the CV which allows for lower CV units to sometimes defeat units with higher CVs.

Goals are used to represent every action that could possibly occur within an RTS game. Types that are found in most games are actions such as attack, defend, explore, recruit, construct, and repair. Each region of a map is assigned an attack goal and each building has a repair goal. These goals are all based on resources such as money or wood or iron. The AI assigns every goal a status in order to keep track of them. Since the AI has to think about and alter the goals of many different actions, it tries to weed out the ones that it doesn't need to think about in order to lighten the extensive demand put on it. Goals that are active mean that they are in the process of being calculated and prioritized. By assigning a goal as active or inactive, it allows the AI to get rid of goals that have no priority. Goals will be assigned a status of inactive if there is no need

for the AI to take it into account, such as a building with no damage will have a repair goal that is inactive. The AI also assigns the status of selected to goals that have been decided to be executed. Finally, a goal will be given the status of finished when the goal has been completed and does not need to be taken into account again.

In order to prioritize goals, the AI will give it a base priority and a current priority. The base priority is the priority of the goal given the current state of the game, while the current priority is similar but takes the resource assignments into account as well as the game state. The priority of goals are constantly being updated in order to keep the AI knowing what to do next. When calculating priorities the AI will calculate the base priority of each of the active goals. It will then assign resources to those goals and rank them from highest to lowest. The AI will then try to optimize the resource allocations and assign resources to each goal in order to optimize the overall goal of the AI. Every goal that has enough resources to make execution possible will then be marked as selected and be put in a queue. The queue will be carried out in order unless the priorities of a goal change again.

A big part of RTS AI is keeping it realistic and human-like. In order to keep gamers content with the gameplay the AI must be unpredictable so the player do not get bored with the same and predictable chain of actions displayed by the computer team. The AI must be adaptive to the game's situation and the player's actions. In order to do this the AI will assign bonuses to certain goals and actions. Tools such as multipliers, minimum priority values, maximum priority values, bonuses, and random bonuses, will all contribute to making the game more random and the actions more human-like. This is called the fuzzy factor and gives a random element to the base priority of every goal. If the fuzzy factor is too low the AI becomes predictable and if the fuzzy factor is too high then it thinks randomly and not intelligently. When generating goal

priorities, the AI will also add a goal inertia to certain goals that prevents the AI from flip flopping between goals with similar priorities. Goal commitment is added in case a goal is selected but there is not enough requirements to be met at the moment. The AI will not spend any resources on lower priority goals until the higher level goal is met. In order to generate the necessary goal priorities, the AI must keep track of the money values and actor values.

The AI needs to understand the values of resources such as gold, wood, iron, and food. It needs to be able to know how much is needed, if more is needed, and of what type is needed. The actor value also plays a large part in the priority of a goal. The value an actor can have varies depending on the type of unit. An actor's value is based on the effect an actor has on the economy, the combat value of the actor, the strategic value of the actor's position, and the location of the AI's territory. For instance, if a unit is in a hotspot or an area that sees a lot of enemy traffic, it will be valued higher. The location of economic buildings also matters in determining values for actors. Economic buildings that are placed in the center of town surrounded by protective defensive structures are more easily defended and have a higher value.

The AI also takes into account build templates. Since sometimes in order to access the most powerful units, certain buildings must be built close to each other. In order to take this into account the AI is given a base priority bonus to build these certain types of buildings next to each other. The build templates also is used by the AI to allocate priorities out to upgrading the city or selling non-matching buildings to make more space in the city. Another big factor, the AI takes into account when generating priorities is an analysis of the map. The map is split into regions where units can move freely. By using a pathfinding system, the AI can calculate the distance between certain points and encourage units to move to goals that are close. The military influence of units also uses map analysis to determine when to attack or stand your ground or

defend. The attack goal is one of the primary priorities that needs to be determined in an RTS game since that is what the outcome of the game all comes down to. The base priority of the attack goal takes into account the maximum number of a player's CV in a certain region and the value of capturable enemy actors would add if they were controlled. The current priority takes into account the ratio of friendly CV to hostile CV, the value of friendly CV in neighboring regions, and whether or not there are known hostile units along the path to attack region. If there are friendly units in neighboring regions, the AI will be more likely to attack since it knows reinforcements can be there to help soon. If there are hostile units along the path to the attack region, then the AI will be less likely to send out their troops to battle since there's a chance that the squad could get distracted or depleted by the enemy.

There are a lot of factors that go into goal-based AI. Every action must be assigned a goal and that goal must be prioritized to determine when it will be performed. The advantages of goal-based AI are it is flexible and powerful, the complexity scales fairly well as AI grows, it can execute more than one goal at a time, and it replicates an emergent behavior (to an extent). The disadvantages of goal-based AI are there are a bunch of calculations that the AI has to take into account, which is nicknamed the "bucket of floats", and the emergent behavior is only emergent to an extent. After a while player can begin to predict the behavior of the AI and the game will lose its entertainment value.

RTS Citizen AI

Citizens are the units in most RTS that can be controlled by the player. They are responsible for the building and repairing of buildings, gathering resources, and the basic unit for beginning the expansion of a civilization. The AI for citizens is usually based on finite state machines and can be found in games such as Rise & Fall: Civilizations at War, Empires: Dawn

of the Modern World, and Empire Earth. The citizens are usually instructed by the player on what to do but then this brings up the question of what to do after the citizens complete the task. The AI tries to explain what citizens will do and how they act depending on certain situations.

The citizens are given high-level goals by the player. The citizens accomplish these goals by following a finite state machine which is a collection of actions and triggers. After given a command by the player, the citizen will continue a certain action in a loop until given another trigger by the player. If no trigger is received by the citizen, the same action will repeat itself.

Citizens are responsible for the construction of a player's buildings. A player will select a citizen or group of citizens and tell them to create a building. When this occurs, the citizens will establish a build site, a location where the citizen is to position themselves in order to build the building, somewhere near the perimeter of the soon-to-be building. Next the AI will use a pathfinding algorithm in order to find a path to the build site. Once at the build site, the citizen will begin to build the building by gradually adding hit points to it. Once the building has been completed, the citizen will go idle and wait for the next trigger from the player. For repairs, the steps are very similar to the building procedure except the building location is already determined. Each hit point that is restored to the building, costs the player certain resources depending on what the type of building requires. If the player runs out of needed resources during a repair, the citizen will go idle and wait for the next command.

When first commanded to construct a building, the AI is responsible for setting up a build site for the citizen. This must be a location around the desired building and must take other citizens' build sites into account before determining theirs. Two citizens can not have the same build site so they must be spaced evenly around the perimeter. A citizen can take a queue of commands from a player. All additional commands are then added to the back of the queue.

Using the queue system, a player can tell a citizen to repair a building, construct a new resource building, then gather a nearby resource and the citizen will then be busy for a while without going idle. The construction process for building is three phases. The first phase is the foundation which must be cleared before construction is to begin. When the foundation is set up, construction does not block paths and citizens can walk through it in order to get to their respective build sites. The next stage is scaffolding and it is the stage during construction. During this stage, the building becomes visible to other players in range and the more citizens working on the building, the faster it is built. Citizens continue to add hit points to the building until it reaches its third and final stage, which is simply the completed stage.

Another action that citizens are responsible for during an RTS game is the gathering of resources. Citizens must gather resources from a resource unit. Examples of resource units could be trees for wood, bushes for food or berries, ore for iron, and rocks for stone. Citizens will collect a resource until their capacity is reached then they will store the resources in a storage unit and return to the resource to continue gathering. As citizens drop off their share of resources, the player's overall resource count will increase, allowing them to allocate them to certain goals such as constructing a building or creating a unit. Unlike constructing a building, a citizen will continue to gather in a loop and will not go idle after completing a building. The action of gathering will only terminate when a citizen can't find a resource or is unable to find a storage unit. At this time the citizen will go idle and wait for another command from the player.

The citizen is a basic and necessary unit needed to advance civilization in most RTS games. The AI behind them tries to let the player delegate actions to each set of citizens then focus on something else. For the most part the AI for citizens makes them self sufficient and prevents them from going idle unless they run out of a resource or do not know what to do next.

RTS Terrain Analysis

When an RTS game is first started, the AI needs to know information about the terrain in order to determine which tactics it should perform. Having knowledge of the terrain of a map can greatly increase the effectiveness and intelligence of an AI system. Some of the things AI will try to determine with the map terrain analysis is where are friendly and hostile troops positioned in contrast to one another, which regions connect to others, and what regions have important strategic value.

When analyzing a map, many AI systems will use a combined approach that mixes a geometric terrain analysis and a pathfinding simulation analysis. The game War Leaders: Clash of Nations specifically uses this technique in order to decipher a map. In the geometric terrain analysis, the AI will use voronoi diagrams using geometric information which is useful in dividing the map into different regions. The voronoi diagram consists of a set of points that are equidistant from two or more distinct obstacles. This information is used to split the map into strictly geometric regions. The map generated by using the geometric approach does not provide information about the strategic value of an area or take into account changing map features. The second process is the pathfinding simulation analysis which attempts to find important paths on the map. It will use a pathfinding technique that determines paths that are travelled the most. It uses the idea that the parts of the maps with a higher concentration of paths are the more important locations. This approach does not provide information about connections or logical areas.

These two processes are then combined to form a map that is able to detect logical areas and their connections and determine the importance of these paths on the map. This map can then be turned into a passability map that shows what regions are passable. The black pixels of

the map represent passable areas and the white pixels represent impassable areas, such as rivers or terrain that is too steep. Each area in a map is classified as a certain type of terrain. Regions are passable areas surrounded by impassable terrain or regions of different type, such as game zones which are broad, open passable areas and chokepoints which are narrow corridors between game zones. Connections are logical links between chokepoints and game zones. Hotspots are the best position to defend a game zone from attacks coming through a particular chokepoint. Hotspots play a large role in the strategy of the AI because it uses these to determine where to position troops and where to invade or defend. Once a passability map is constructed, a process called noise reduction is used to clean the map and remove small pockets of impassable terrain the provide useless divisions in the data.

The clean passability map is now ready for the iterative image process which will eventually output an image on which each region is clearly marked by different colors. By using the process known as iterative image processing, the passability map is manipulated until passable terrain, impassable terrain, game zones, and chokepoints are all clearly labelled by different colors. The AI system can use this knowledge to help develop a strategy that can improve its chances against the enemy players. The AI can determine the properties of each region, which areas contain allied and enemy units, hotspots: which connections must be defended or attacked depending on enemy position, direction of hotspot: where to place units at appropriate locations relative to the connection, and weighting information: which regions and connections are most important. Using this vast amount of knowledge allows the AI to make effective moves to counteract the player's actions and make the most intellectual in-game decisions. Without the use of terrain analysis, the AI would be blind as to where and when to attack or defend and would be at a great disadvantage.