ARTIFICIAL INTELLIGENCE IN VIDEOGAMES

PRACTICE PROJECT

Realizado por:
JACINTO ARIAS MARTÍNEZ
ADRIÁN SÁNCHEZ LÓPEZ

May, 2012

Abstract

Contents

1 Project definition and design principles

1.1 Game definition

The game we are proposing is an RTS game in which you will be managing a large horde of units in order to break down a enemy fortress.

The main feature of this game is that you won't be able to directly control your units, in case of that, you will give them orders and the units should behave according to them but with a high free will grade.

The horde will be separated into groups of units that you will be able to select. Using the mouse and the graphic interface provided, you will be able to control a group of unit at once, giving orders to your troops should alter its behaviour and they will follow your order in their way.

There will be computer controlled enemies that will kill your units in order to defeat you. A longer description will be included in the next section.

1.2 Motivation

We have chosen this kind of game because we think that the AI integration possibilities are huge. AI principles can be integrated in many of the game components, for example, in the game engine, AI techniques can be used in order to define and implement the units free will, also it can be combined with an agent-based approach. AI could also be applied to make opponents for the game, we should define their strategies, decision, etc...

1.3 Game classification

The presented game could be classified as:

- **Game type:** Real Time Strategy
- **Type of use:** The machine is controlling several characteristics as physics, ground generation and playing as an opponent.
- **Theme:** Horde of units.
- **User age:** No children are allowed to play this game because of the violence it contains.
- **Skills development:** In this game player will develop spatial abilities and also reasoning ones by controlling the units and thinking fast strategies in order to win the game.
- **Platform:** Originally it is developed for PC, with compatibility with Microsoft Windows and Linux.
- **Technology development:** All the game has been developed using free software: Ogre 3D, Blender, Gimp, OIS and OpenAl.
- **Development company:** Jacinto y Adrián ó Adrian y Jacinto Developing.
- **Year of development:** 2012.

2 Artificial intelligence principles I: Computational Theory

2.1 Game elements

In this game your goal is to conquer the centre of the map by getting a large number of zombies inside the enemy fortress.

There are four main elements on the game:

- **Zombies:** This is the interactive element of the game. You will control a few groups of zombies that will start spread over the map. You can order to each group of zombies move to a point, but if they are hungry they won't obey you and will wander randomly. If they can reach the goal (centre of the map) they should attack the enemy, but beware because they can shoot and kill the zombies.
- **Food:** Spread along the map you can also find food, food is necessary for the zombies to get strong and obey your orders. If a group of zombies find food they will eat it.
- **Enemies:** In the centre of the map, and spread along it, you will find a bunch of enemies. They will be patrolling all the time and trying to kill your zombies if they come close enough. Your goal is to kill every enemy on the map (or reach the goal).
- **Goal:** The goal of the game is to catch the centre goal, a big Brain. If a zombie touch the Brain, you will win; but is not too easy because all the enemies are patrolling and protecting the goal.

You can also find obstacles and other passive elements along the map, such as plants, trees, rocks .... That will make your zombies harder to move in a pack.

2.2 Knowledge extraction

The basic strategy a user has to know about how to play this game is that everything has to be accomplished by using a large number of units. Some enemies would be tougher than others so you should take a look of the map before start to send all your units to an early end.

When you plan your attack watch for fixed turrets or enemies in far positions in order to avoid them and don't waste troops killing them. Also, coordinate your units to attack at once and corner the stronger enemies to quickly take them down before they cause so many casualties.

The food should be a secondary goal, but you should collect many of them in order to make your army stronger for the last assault if you want your plan to be committed. Otherwise, your troops will fall into chaos and wander around the map letting the enemies to slowly kill them.

The game is simple, so when you discover how the different enemies behave you should be able to anticipate you plan by taken a quick look of the map disposal.

2.3 Artificial intelligence applications

In this section we will identify some of the problems that can be solved using artificial intelligence solutions.

The most immediate problem to solve is the behaviour of the units. With most of the techniques we should solve problems like units movement, path finding, collision avoidance or unit tracking.

Other aspect of the game that can be involved in AI developing can be the opponent strategy. We can implements such methods in order to give the opponent more intelligence making the game most interesting. For example we can apply these techniques for choosing the army or placing in on the map.

2.3.1 Movement problems

When speaking about movement we have to difference between zombies and enemies, because their movement models are not the same.

The zombies model is based in two parameters, firs of all we have the destination point which is the zombies goal to reach, on the other hand we have the hunger of the zombies group. If the zombies are hungry they would tend to wander around while reaching the point, if the hunger level gets low the zombies will appear to come closer to the point obeying the player's order directly.

When programming the zombies movement we should take those parameters in consideration and apply an straight movement and a movement pattern for randomness when wandering. These movement pattern should be applied by using AI techniques, with different techniques we could achieve richer patterns going from a random pattern to behaviours like birds flocks, particle swarm or something different.

A rules based system, a neural network or CBR can be implemented to achieve this.

2.3.2 Path finding and collision avoidance

This problem is mostly a feature of the game engine. With the integration of a physics module which will simulate the collisions of solid models there should be physical interaction between the elements of the game. For example, a unit will not be able to pass through a tree or a wall or also through another unit; also, it will have to adapt its movement to the terrain topology.

For that reason, when a element is commanded to move through the map the game engine should find for him a valid path in order to keep the movement realistic and avoiding that element (normally and avatar) to get stuck in somewhere.

Normally, those path finding problems can be solved applying heuristic search techniques like the A star algorithm.

2.3.3 Opponent strategy

At first, the game is being designed as a single player one. For that reason an AI should be designed and programmed in order to control the opponent.

This AI system should variate in complexity, different models and behaviours can be programmed in order to give the game a different approach. For example, a simple rule based system can be implemented in order to do that conferring the an "arcade" approach. However, we can also integrate an combination of different AI techniques even including learning, to make a robust opponent for a different kind of game with also a really good academic interest.

Independently of the selected system, the problems that should be solved by it are the same:

- Moving and positioning the units.
- Look for, select and attack enemies.

Those problems can be solved using a very simple system or using a high-level complicated strategy.

To commit this task, we provide a framework in order to identify a limited number of elements to work with:

- **Enemy behaviour:** There are two behaviours in which the enemy can be at a given moment:
  - Patrol: The enemy is moving around the map, moving to a given point that the punctual movement provider tells it.
  - Seek: The enemy is rotating over himself at a constant speed, can be done for searching enemies in range.
- **Enemy aggressiveness:** The enemy can be in an aggressive mood or not. If aggressiveness is on, the enemy would shoot at anything that comes into range.

- **Environment data:** The enemy can know at any moment about its own data and the zombies data over the map with no restriction.

Additionally to those elements, each unit has different parameters, that would make it stronger o weaker:

- **DPS: Damage Per Second:** Is the amount of damage that an enemy deal to the zombies.
- **Life:** The amount of damage that an enemy can be dealt by zombies before die.
- **Range:** The maximum distance that a unit cat attack.
- **Speed:** The velocity of the enemy displacements around the map.

Using all of these elements we should implement all of the different AI for the different kind of enemies in the game. In further sections we will explain about the different implementations.

3 Artificial intelligence principles II: Representation and algorithms

3.1 Application of search techniques

3.1.1 Path finding

As we said above, we can apply those techniques in order to find correct paths for the units to move. In order to solve the problem we will use the popular A star algorithm, and for that we have defined the next representation scheme:

- **Input space:** As an input the algorithm will receive the game's map divided in regions and represented as a non directed graph where each node will be either a passable or an impassable region, the actual location of the unit to move and the final location where the unit want to reach. The arcs of the graph will be weighted with the distance between regions and only the nodes which represents adjacent region will be connected.
- **Output space:** As an output the algorithm will return a valid path of the graph, directly interpretable by the game engine.
- **States:** An stated will be conformed by a node on the graph. Only the nodes that are passable can be considered as valid states.
- **Initial state – Final state:** The initial state corresponds to the node which represents the region of the map where the unit is located.
- **Operators:** Ramification through states should be done by moving directly to the neighbours of the current node.
- **Function of heuristic evaluation:** The heuristic function will just be the cost of the path of going directly from the origin to the destination without considering the condition of the regions. The distances will be precalculated for the algorithm.

3.2 Application of rule based systems

3.2.1 Movement model: Simulation of a birds flock

As we said, we can use a RBS in order to provide a model of movement for our units. In this case it will be a simple one with a really small set of rules but when applied to a group of units it is fully functional and provides emergent behaviours so this could also be seen as an **artificial life** system.

For the implementation which will be explained later we have defined the next representation scheme:

- **Input space:** As an input space we provide as antecedents the unit actual location, and the other units actual location.
- **Output space:** As an output space the system returns a destination location where the unit will go as the movement module expect.
- **Input and output representation:** The locations are represented as 2D points in a cartesian system, so we can make some calculations on them.
- **Statements base:** Nearest unit distance.
- **Set of rules (knowledge injected):** The set of rules consists of three of them, enough to provide an approximation for the birds flock behaviour. The rules are the following, ordered by highest priority:
  - *If you are far away from other birds, head toward the nearest bird.*
  - *If you are about to crash into another bird, turn around.*
  - *Otherwise, fly in the same direction as the bird next to you.*
- **Considerations for the inference engine:** As this will be a really short inference cycle, the inference engine will be emulated using sequential conditional and the statement base using numerical variables.
- **¿Learning?:** As this system is based on purely injected knowledge it will be difficult to include any learning. Perhaps the distance parameters of that appears on the rules should be later optimized.

3.2.2 AI behaviour: The hunter

By using a simple set of rules, we can make a very effective opponent using a really low resources consuming technique.

For the implementation which will be explained later we have defined the next representation scheme:

- **Input space:** As an input space we provide as antecedents the unit actual location, and the zombies locations.
- **Output space:** As an output space the system returns the next action to achieve, that could be move to a given point or to stay and seek for enemies.
- **Input and output representation:** The locations are represented as 2D points in a cartesian system, so we can make some calculations on them, the output is given directly to the program to interpret it.
- **Statements base:** Previously attacked unit, nearest unit location.
- **Set of rules (knowledge injected):**
  - *If the nearer enemy is too close move away before attacking it.*
  - *If an enemy gets in range, shoot at it.*
  - *If you has just killed and enemy, stay and seek for the rest of the pack*
  - *If you don't have an objetive move closer to the nearer enemy.*
- **Considerations for the inference engine:** As this will be a really short inference cycle, the inference engine will be emulated using sequential conditional and the statement base using numerical variables.
- **¿Learning?:** As this system is based on purely injected knowledge it will be difficult to include any learning. Perhaps the distance parameters of that appears on the rules should be later optimized.

3.3 Not implemented techniques

The next techniques don't have a formalized representation scheme for our game, but they have been discussed and proposed for typical usages inside this game.

The information provided next consists in factional ideas that could be revised and implemented in a more mature stage of development of our game. By the moment, we provide this information as an exercise for the current subject.

3.3.1 Application of CBR

The idea is to use CBR in order to define some strategies for the opponent, giving more flexibility than a RBS. We have different strategies for the opponent, and we need to choose one. The possible strategies are: **patrol**, **explore**, **attack to zombie detected** or **protect the Brain**.

The Case-based reasoning has been formalized as the following a four-step processes:

- **Select:** Given a target problem, retrieve from memory cases relevant to solving it. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived.

  For our problem of select the correct strategy, we retrieve from our database the most case relevant (with best solution) that solve the choice with the actual status.

- **Matching** Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation.

  Adapt the retrieve solution to include all the particularities of actual situation, such as different number of zombies, positions, and so on.

- **Review** Having mapped the previous solution to the target situation, test the new solution and, if necessary, revise.

  We need to check, if after adapt the retrieve solution of strategy selection, this is a valid solution to the problem or we can not solve the choice. If a solution do not give good results (is not valid), we need to make a revision, such as modify the selection trying to do a better solution.

- **Update** After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory.

  When we have a new successfull selection of a strategy, we include it to out database like a new case for future steps.

The point of using this kind of techniques is that we can but it in the game and face it against human players to see how it evolves.

A good variation of this should be using it to adjust the selected strategies of each enemy in the map. It could be used as a **meta-algorithm** to do that, designing and initial configuration of the AI system by assigning a pre-made behavior to each unit and evaluating the result at the end of the game.

3.3.2 Application of connectionism

The idea of connectionism is really similar to the previous one.

By using the different states of the enemies, and the enviromental information we can design a multi-layer neural network which outputs would be the next behaviour that the enemy should get to practice.

The big problem with this kind of model is the need of learning. With the CBR approach we would be able to put it in the game and teach it by playing with human players, but the neural network learning process is more complex and requires more simulation that would be really difficult to make without the intervention of an human player.

3.3.3 Evolutionary computation

3.4 Mathematical approaches and other algorithms

As this game is a simple one, most of evaluated problems can be and have been solved by using more simple solution than using complex AI techniques.

Basic algorithms, and mathematical approaches have been applied in some situations that we are going to explain next.

3.4.1 Collision system and geometry navigation

For physics, like collisions, we've implemented mathematical solution based on raytracing. Those solutions are based in linear algebra and analitic geometry. So mathematical equations had been implemented directly in the code in order to solve the problem of collisions or for example, calculating the range of fire for the enemies weapons.

Also mathematical equations have been implemented in order to manage the navigation through the map. Elemental trigonometry and analitic geometry has been used in order to provide the mathematical support to spatial navigation.

3.4.2 Randomness

Randomness is deeply used in this game. For a lot of situations a random number is able to solve a problem and also to add some richer properties to the game, making it more stochastic and so amusing for the players.

3.4.3 Directly applied knowledge

In some situations, it could be a good point to directly apply some of the knowledge extracted directly by using simple programming techniques.

Although we could try to formalize some of this implementations by using some of the previous techniques, most of them are too simple for that. So state machines, search algorithms and other basic programming techniques can be found deeply used in some of the AI modules and can be considered as cornerstones.

3.5 Cooperative behaviour

Cooperative behaviour should be a good point to add to this game.

We just have to add the other enemies information to the environmental information they get and use it properly. We have two ways of doing this:

- **Define a communication protocol:** By establishing a direct communication between the enemies they could use joint strategies in order to win the match to the player. Movement formations or spread themselves along the map to cover more area.
- **Define an overlord controller:** This kind of controller could make more meta-game level strategies like a human player. And so the extracted knowledge could be applied here.

Despite of the selected technique, cooperative behaviour an agent based approach would enhance the emergent behaviours of the units in the game, giving it a more "intelligence" appearance.

4 Artificial intelligence principles III: Implementation

4.1 Software engineering and game engine implementation issues

As this subject is not focused on videogames programming, this section will cover the basis of the implementation in order to identify and follow the different elements of the game engine. We are going the provide the fewer diagrams and models needed to understand the structure and the function of this application.

4.2 Programing with Ogre3D

OGRE (*Object-Oriented Graphics Rendering Engine*) is a scene-oriented, flexible 3D rendering engine (as opposed to a game engine) written in **C++** designed to make it easier and intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics. The class library abstracts the details of using the underlying system libraries like *Direct3D* and *OpenGL* and provides an interface based on world objects and other high level classes.

The learning curve of OGRE is very high, it costs a lot to adapt and understand all the systems and methods for developing a first application.

But on the other hand, once you have come to understand its operation, continue to develop and incorporate new functionality is much more fluid. But how OGRE is only a rendering engine, we need other kind of software to desing our models and animation. We have choose Blender for 3D modeling.

4.3 Artwork in Blender and Gimp

Blender is a free and open-source 3D computer graphics software product used for creating animated films, visual effects, interactive 3D applications or video games. Blender's features include 3D modeling, UV unwrapping, texturing, rigging and skinning, fluid and smoke simulation, particle simulation, animating, rendering, video editing and compositing. It also features a built-in game engine.

GIMP (GNU Image Manipulation Program) is a free and open source software image retouching and editing tool. Gimp has tools used for image retouching and editing, free-form drawing, resizing, cropping, photo-montages (combining multiple images), converting between different image formats, and more specialised tasks.

As Blender is just a 3D modeling software, our models and animations need to be exported to use it on OGRE aplication. First of all, we desing the 3D geometry of, for example, a zombie in Blender; then, include some textures and details with an external 2D editing software (like GIMP), and finally export our model to a OGRE format.

4.4 AI techniques integration

In this section we will talk about the structure of the game engine and how we should integrate the different techniques in order to provide not only a videogame but also a modular platform in which we can test the different approaches seen in class and study its behaviours.

On the one hand and as the game engine, we are going to introduce the different elements that we should integrate to provide a videogame interface and its elements. On the other hand as a platform for testing AI techniques we are going to let an empty place in which we can plug in the AI techniques as different modules implementing an specific model; for example, we are going to implement the hordes units movement in the game engine letting a place to plug in a model, which can be a random model or a richer approach like a rule based system or a neural network.

4.4.1 Movement module and models

The movement models for the units will affect to their free will movements during the game. Providing richer model will make the make more interesting as our avatar should behave in a more complex way and also resulting on situations of emergency.

The movement module itself should catch some of the parameters from the units and environment and provide a function *calculateMove()* that, for a given avatar, will calculate the next move.

The next models have been included in the game and all fully interchangeable without changing anything on the code rather than its name:

- **Random model:** As it name says it model should give a random movement to each unit that ask for it. Normally units following this model would move on any direction without taking care of anything else (environment or other units).
- **Bird Flock:** This model is based on a biological approach and has been implemented by using a ruled based system, the specification is shown before in its own section. With this model the whole group will maintain some coherence and will move along together, it neither has any consideration on the environment.

4.4.2 Behaving and action module and models

4.4.3 Strategy and meta-game module and models

5 Conclusions

5.1 AI Results

We have obtained the following results, in terms of artificial intelligence behavior: TO DO

5.2 Development costs

The approximate time invested in the development of the application has been: TO DO

5.3 Showing

A sample screenshots of the application Zproyect: TO DO