Computer Science and Information Systems 20(2):765–792 https://doi.org/10.2298/CSIS220719021Z

Using Artificial Intelligence Assistant Technology to Develop Animation Games on IoT

Rong Zhang

School of Artificial Intelligence, Dongguan Polytechnic,
Dongguan, 523808, China
443798430@qq.com

Abstract. This research proposes an XNA animation game system with AI technology for action animation games in mobile devices, based on an object-oriented modular concept. The animation game function with AI technology is encapsulated into independent objects, through the combination of objects to build repetition. It adds AI technology to the finite state machine, fuzzy state machine and neural network and attempts to combine the traditional rule-base system and learning adaptation system to increase the learning ability of traditional AI roles. The main contributions are compared with traditional methods and the AI animation game system is shown to have more reusability, design flexibility and expansibility of its AI system through the object composition approach. It adds AI technology to combine the traditional rule-base system and learning adaptation system to increase the learning ability of traditional AI roles. Therefore, AI animation game producers can accelerate their processes of developing animation games and reducing costs.

Keywords: Artificial intelligence assistant, Animation games, Neural network, Object-oriented modular, Fuzzy state machine.

1. Introduction

Due to the rapid development of the global digital content industry, various countries have invested in the promotion of the digital content industry, and the animation game industry has become one of the most promising industries. The research pointed out that in terms of annual expenditure, the growth rate of television (TV) animation games is 9.6% higher than that of TV and is second only to the Internet. In terms of the average hours spent per year, TV animation games are growing and the fastest-growing media has a compound annual growth rate of 7%. Animation game producers develop games with AI technology, increasing the perception of players that they are intelligent beings, so that players can have a deeper and more interesting experience when interacting with these AI characters in animation games. From this point of view, AI animation games are not only an indispensable technology for most animation games but are also one of the sources of interactivity and fun in animation games [1, 2].

In today's digital electronic technology, the technical power of the digital animation gaming industry is growing rapidly, through the mutual promotion of software and hardware. The speed of the improvement of animation game graphics is obvious in all

game fields. In the early 2D surface to the mid-term 3D stereo, and to the next generation of high-resolution images, animation games are constantly progressing in terms of screen effects that continue to shock players. AI research experts claim that when animation game screens improve, animation games are not promoted in the same way [3, 4]. However, due to the advancement of graphics technology, animation game characters have smoother movements, more realistic textures and more complex expressions. If they do not have an effective AI animation game to control their behaviour, the animation game characters will be deemed as not having a real appearance, and instead will be characterized as having asymmetric and strange behaviour [3, 4]. Therefore, as regards superior animation games, AI is a very important technology for today's animation games. In the past, animation game development used DirectX for programming, however, DirectX did not provide much of a design function to animation games, therefore, the designer was responsible for a large number of implemented parts. The XNA animation game is an integrated cross-platform development environment for PC/XBOX mobile animation games and includes an animation game development library, packaged software tools and various other development tools, so as to provide the most convenient and efficient development environment for AI animation game developers. It brings together 3D drawings, physics engines and animation game production process management, as well as resource management and other applications to help AI animation game producers accelerate the development of animation games and reduce costs. Fig. 1 is a diagram showing the relationship between XNA and animation games. In Fig. 1, XNA is an animation game function library, which provides the basic functions of animation games and an empty animation game architecture. The designer only needs to expand the functions according to its architecture [5, 6]. The second layer is the animation game engine, designed by the designer to expand and establish the basic category library and functions, according to the functions library required by the animation game. The third layer is the animation game code, using the first and second layer category libraries, and the function is used to write the code for the actual operation of the animation game. In the case of a small animation game, the designer can even skip the second layer, as long as the function provided by XNA is used directly to write the animation game program in an empty animation game; this can make the animation game work. Based on the aforementioned advantages, XNA provides a very good development environment for amateur animation game designers and students [7, 8].

The related AI animation game technology research can be roughly divided into: chess AI animation game implementation, AI animation game implementation, based on the open architecture of existing animation games and self-made AI animation game system implementation [9, 10]. Most AI animation game technology researchers will choose animation games with simpler rules to realize the idea of AI technology. Nowadays, many commercial animation games openly authorize players to modify part of the animation game program. Players can implement their own ideas in the animation game, according to the open modification framework of the game. This has also led to research such as the AI implementation of animation games, based on the open architecture of existing animation games [11.12]. This research uses open-licensed commercial animation games to modify programs, so as to realize researchers' AI technical ideas, although researchers can quickly use such animation games to implement AI animation games. However, these will also be subject to the original

architecture of animation games, therefore, the AI technology that can be implemented will be limited. Although the AI system implementation of self-made animation games needs to build an animation game system independently, it gives researchers considerable freedom and flexibility. The first objective of this research is twofold. Firstly, the aim is to implement action animation games with XNA in the AI system. Through the concept of object-oriented modular objects, an AI system is built with reusability, design flexibility and expandability. In addition, finite state machines, fuzzy state machines and neural network AI technology applications are added, combined with traditional rule-based systems and learning adaptation systems, so that traditional AI roles can increase their learning capabilities [13, 14]. The second purpose is to show that XNA can simplify the characteristics of animation game design through the process of implementing the AI system; it can explain and record the implementation process to provide a reference for related researchers and producers. This research sets the implementation goal as the AI system implementation of action animation games. By focusing on the AI system implementation of a single animation game type, the results of the research purpose can be achieved within the expected timeframe [15, 16].

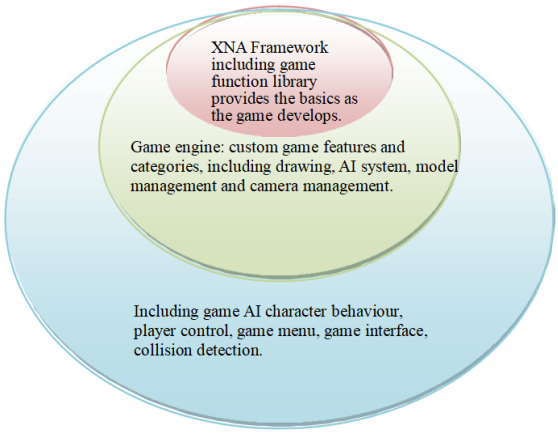

Fig. 1 The relationship between XNA and animation games

Although the AI animation game is implemented using an academic concept, issues, such as animation game execution efficiency must also be considered. Therefore, an AI animation game usually simplifies the complex algorithmic process or only uses its concept to implement an AI system, as its main purpose is to perform the desired AI effect under the allowable performance consumption. This research will focus on the AI system used in animation games, with the aim of demonstrating that the research results can be more in line with the current animation game production flow process, so as to improve the performance value of the research results. Animation game production can be roughly divided into three areas: planning, art and programming. Each area constitutes a different professional field, however, this research will only discuss the programming element of AI animation game production [17, 18].

The first section discusses the research background and motivation, as well as the research purpose and scope of the research. Section two discusses related background

knowledge and the AI performance that is emphasized in the basic types of animation games. Section three focuses on the system planning and analysis of AI animation games and explains the system architecture. Section four considers the implementation of AI animation game systems and explains in detail the function and structure of each object. Section five discusses the implementation, which through combination and connection becomes a working example implementation; this section also highlights the reusability, design flexibility and scalability of the AI system. Section six comprises the conclusion and explains the research results and the future work.

2. Related Work

2.1. Animation Game Types and AI Performance

The type of animation game greatly affects the function and direction planning of the AI system. Different types of animation games focus on different AI performances. Therefore, AI systems are rarely universal and are usually built for specific animation games. There are many types of animation games today. Table 1 lists the common basic animation game types. In addition to the basic types in the table, there are also many mixed types, such as the Action Role Playing Game, ARPG) and the Action Adventure Game (AAVG), [19, 20].

The basic animation game types listed are based on these animation game types. The following key AI performances are summarized [21-25]:

1. Fixed behaviour: the behaviour of a fixed mode does not change according to the time of the animation game. The designer defines a number of behaviour patterns and conditions that trigger behaviours and the AI animation game characters make corresponding behaviour patterns, based on the defined conditions. They do not change in line with the time of the animation game or any other factors.

2. Variable behaviour: the behaviour of the non-fixed mode will change in accordance with the time of the animation game or other factors. The designer also defines several behaviour patterns, but the conditions that trigger the behaviour will change, according to the time of the animation game and other factors.

3. Simulation behaviour: AI characters record and simulate the player's good performance in animation games.

4. Group action: this is the behaviour pattern of AI characters in group actions of multiple units. For example, in a real-time, strategy animation game, when a large number of AI animation game units are moving, the group action will form a melee unit in front and a long-distance unit behind or teamwork behaviour in basketball or football animation games.

5. Strategic thinking: this constitutes AI management performance that belongs to high-level decision makers and supervises and collects various information in animation games at a higher position. It integrates the information to make decisions and informs single or multiple AI animation game units of the decision.Nowadays, the number of animation games is quite considerable and each animation game may contain multiple

AI performances. Therefore, Table 2 mainly focuses on the most common AI performances of this type. It shows that the AI performance of action animation games has only one fixed behaviour, although role-playing animation games and adventure animation games are also in the same situation. The difference is that the objectives of the other three types of animation games are not only focused on defeating enemies. For example, role-playing animation games also focus on character training and adventure animation games focus on solving puzzles, dodge mechanisms and the process of jumping on to a platform [26-30]. The goal of the action animation game is to defeat the enemy. Therefore, this research aims to enhance the player experience of action animation games, by enriching the reactions and actions of AI characters. In order to achieve this goal, the research will focus on AI characters to add changing behaviour. The AI animation game is the broadest definition, including everything from simple chasing and dodging movement modes to neural networks and genetic algorithms. Using AI to allow non-player characters to show different personality traits or to present human-specific emotions and tempers, is one of the options that can be considered in the design of animation games [31-35]. No matter what it entails, as long as it can give people a certain level of wisdom, make animation games more addictive, more challenging and importantly fun, it can be regarded as an AI animation game.

Table 1. Common types of animation games available currently

Type of animation game	Feature description	Famous animation
Role Playing Game	Focuses on plot description and character development. Usually has a broad, complete world view and gives attention to the player's sense of investment in the world of animation and game.	Final Fantasy[12]
Adventure Game	Comprises a puzzle element and players must solve through the puzzle to pass through the levels.	Tomb Raider [6]
Real-Time Strategy	Players need to control a large number of animation and game units at the same time to fight, and the animation and game unit action is immediate.	Age of Empires[11]
First-Person Shooter	A first-person perspective of the animation game; the players shoot as the main means of attack.	Call of Duty
Platform Game	There are a lot of units and organs on the map. Players must jump to avoid the organs, so as to play the animation game.	Super Mario [15]
Sports Game	Simulation of various sports, such as ball games, swimming, skiing, etc.	NBA Live
Racing Game	This game has a sense of speed, such as racing cars or racing boats.	Need for Speed
Action Game	A single character completes a level by knocking down the enemy. The action is instantaneous.	Ninja Gaiden
Fighting Game	Fights a single enemy at a time, using a variety of knockouts.	Street Fighter
Table Game, TAB	Realistic desktop animation game simulation, such as chess, monopoly, Mahjong, etc.	Monopoly

2.2. AI Animation Game

AI animation games are divided into three categories, namely rule-based systems, goal-based systems and learning and adaptation systems. The following relates to the rule-based system and the learning and adaptation system, which are more commonly used in the development of animation games.

1. Rule-based system: this type of system has a long history in animation games and also uses the most common AI technology in animation game development. Its operating principle involves the designer defining the conditional rules corresponding to the behaviour of AI characters in advance, then using the programming language The conditional judgement (if, then, else, switch, case) is used to control the behaviour of AI characters. The advantage of this type of system is that it is simple and easy to understand as well as to debug and implement, however, its disadvantage is that it lacks learning ability and is easily predicted by players. There are three common architectures for such systems, namely the finite state machine (FSM), fuzzy state machine (FuSM) and production systems.

2. Learning and adaptation system: as the name suggests, this type of system has the ability to learn and adapt. It can make AI characters more changeable and challenging in animation games since players cannot easily predict the behaviour of AI characters. Due to the continuous improvement of AI characters, the entertainment life of animation games has also been extended. According to the AI learning technology used, this can roughly be divided into two categories, namely indirect learning and direct learning.

1. Indirect learning: indirect learning is a mathematical statistical method combined with a rule-based system. It counts the player's control patterns and habits, and analyses them, then uses the analysed data to modify the AI character's behavioural rules, so that the AI character has the ability to learn and adapt to the player [36, 40].

2. Direct learning: direct learning is based on AI technology with learning concept algorithms as the system architecture, allowing AI characters to have autonomous learning capabilities through learning algorithms. In this type of method, the designer usually establishes a learning system first, then creates a system with learning capabilities through different training methods. AI characters in animation games use the trained system to conduct behavioural, decision-making analysis. An AI character, built entirely using this type of system has strong learning and adaptability capabilities, but this will also cause problems for animation game developers, such as difficulty in debugging or the AI may deviate from the expected effect. Moreover, such systems, which usually contain complex algorithms are difficult to understand and implement. There are two common applications of learning and adaptation systems in animation games, namely the artificial neural network and the genetic algorithm [41-45].

Table 2 below compares the advantages and disadvantages of the two AI systems and shows that the advantages and disadvantages of the two AI systems have complementary effects. If only one single type of AI system is used in an animation game, when the animation game is scaled, the larger the size, the greater the number of shortcomings. Therefore, in current animation game development, different functions are usually established for different needs. Either a subsystem of power is established or the AI work is divided into several small systems, then multiple subsystems are combined to form a complete AI animation game system, so as to maximize the advantages and minimize the shortcomings. Consequently, the AI system modularization of the key

technology of the AI technology is very important. The modularized AI technology will be reusable in the system and can effectively combine the subsystems. The meaning of modularization is reusability, which can improve flexibility [45].

Table 2. Comparison of the advantages and disadvantages of the rule-based system and the learning and adapting system

	Advantage	Disadvantage
Rule-based system	• Easy to understand. • Easy to implement. • Debugging is easy.	• The larger the animation game scale, the more the program volume will lead to confusion. • Predictable.
Learning and adaptation system	• It is learned. • It is hard to predict. • It simplifies complex programming.	• Implementation is not easy. • Debugging is difficulty.

2.3. FSM

FSM: also known as finite state automata or state machine for short, this is a mathematical model that represents a finite number of states and behaviours, such as transitions and actions between these states. A state machine usually contains three elements: the transition functions between all states, inputs and connection states in the state machine [21-25]. The FSM is used in the AI program of animation games. The AI chasing the player's enemy in the animation game is the application of the FSM and this is used in the AI of animation games. The reason for this is that it easy to understand, implement and debug characteristics, and its characteristics also make finite states. In the process of developing animation games, the machine often has outstanding performance. In the following section, the enemy AI in the classic animation game, Pac Man, is used as an example to explain the use of FSMs in animation games [46-50].

The FuSM is actually an extended variant of the FSM. These statements are closer to everyday problems and semantic statements, and examples are used to illustrate the concept of fuzzy logic. In an animation game, there is a castle and according to the concept of classic logic, when the character enters the castle, it is 1 and not in the castle. The inner time is 0, which makes the character's behaviour in the animation game more flexible and realistic. The difference between the FuSM and the basic state machine is that the FuSM allows multiple states to run at the same time, while the basic state machine can only run one state at a time (Kwon and Shin, 2005). The parameter can be used as the stimulus value of the FuSM state. Firstly, the stimulus value standard for triggering each state, when the enemy is close to the point, may result in the FuSM operating in the three states of moving, shooting and dodging at the same time.

The production system can also be called an expert system and combines the functions of a knowledge database, rule management and decision-making. It is usually used to solve expert problems in a specific field. In the production system used by AI characters in animation games, the users of the system are the AI characters and the process of operating the system used by the AI characters is automatically run by the

computer, so there is no need to provide a user interface to the AI characters. Animation game programmers usually write scripting languages corresponding to the AI system and provide these to animation game planners to design AI animation games [26-29]. The production system is famously used in animation games and the main AI system is established by the production system. The computer AI in animation games can learn the building order and the resource requirements of the building through the knowledge base, then use the reasoning mechanism to determine the search for resources and collection acts [51-54].

3. Research Design

3.1. AI System Construction Process

The scope of the AI animation game is quite large; complex human thinking simulation to path search are all within the scope of the AI animation game. Therefore, before building an AI system, one must first determine the main use of the AI system, such as purpose, expected effects or information relating to animation games, then the function and architecture of the AI system can be planned, based on the established information content. Finally, the system can be implemented based on the planned content. The AI system construction process can be divided into three periods, the system direction and basic content setting period, the AI system planning period and the AI system implementation period:

1. System direction and basic content setting period: the key function of the AI system builders is to confirm the animation game type, animation game content and the expected performance goals of the plan, as well as to carry out the preliminary planning of the system, based on the information obtained and to analyse and select the required software.

2. AI system planning period: the main work content is to carry out detailed planning of the AI system, including the AI system architecture, the application scope of AI technology, the information required by the AI system, the operation method of the AI system and the animation games to connect to the main structure.

3. AI system implementation period: this relates to the AI system construction work, according to the content planned in the AI system planning period, and the results are realized by program software. In addition to the implementation of the system, this issue also includes system testing and problem modification.

3.2. Software and Hardware Requirements

(1). Software: Visual C# was developed to realize the most efficient programming language in the .NET Framework. Visual C#, like Java, is an intermediate code language. The base language used by XNA is C#, so the main language used in animation game programming is Visual C#.

(2). Hardware: the software used in this study is XNA Game Studio. Therefore, based on the hardware requirements of XNA, it is recommended to have at least 4GB of physical memory and a CPU clock of 2.0GHz or more.

3.3. AI Animation Game System Design

The AI animation game system is a subsystem built on animation game architecture. The AI system cannot operate independently because the AI system itself does not include graphics, model management, cameras and other animation game functions. It must be combined with other units of the animation game architecture. Therefore, for the AI system to function normally, it must have a foundation architecture in the main body of the animation game. It uses XNA as the animation game development platform, therefore, XNA will be used as the bottom layer of the animation game's main structure for planning purposes. When creating XNA's new animation game project, XNA will provide designers with an empty animation game architecture and designers must follow this architecture to expand and extend animation games. Fig. 2 shows that this architecture contains five main functions, which are Initialize, LoadContent, Update, Draw, UnloadContent, when the animation game starts; Initialize and LoadContent will be executed in sequence. After running Update and Draw, these two functions will be executed repeatedly during the animation game process until the animation game ends and UnloadContent will only be executed before the end of the animation game. The five functions will be introduced as follows:

1. Initialize: The first part executed when the animation game starts, the function is to initialize various animation game data.
2. LoadContent: this function is used to load the external data required by various animation games, such as animation game models, text files, pictures, videos, etc.
3. Update: the main loop of the animation game will be continuously executed during the animation game process to update the content and information of various animation games.
4. Draw: similar to Update, Draw will be executed continuously during the animation game process and its function is to update the animation game screen.
5. UnloadContent: this is used to release the memory space used before the animation game ends.

XNA provides a sample program function, which allows designers to quickly create an object that includes the infrastructure; the operation process of this object is also the same as the main structure of the animation game. Fig. 3 is an animation game architecture diagram, established in accordance with the animation game functions required by the AI animation game system. The function descriptions of each object form part of the animation game architecture.

3.4. The Concept Architecture of the AI Animation Game System

AI is a computer technology that simulates human thinking reactions, judgement logic, and learning ability, since the purpose of AI animation games is to provide animation

game characters with a vivid performance, allowing players to believe that the characters are alive, enhancing the animation game play and challenge of animation games and making it easier for players to become absorbed in the animation game content. At the same time, animation game development must also consider its performance. A poor performance of animation games will affect players' acceptance of animation games. The AI animation game system architecture divides the characteristics of AI into the following four key points, so as to simulate human thinking behaviours: sensation, memory, thinking and judgement, action. These four key points will be explained as follows:

1. Sensation: this is referred to as information reception and represents the feelings of animation game characters in relation to the animation game world, for example, sight or hearing.

2. Memory: this is known as information access and represents the information that an animation game character must record in the animation game world, such as player information, including player level and equipment or the attack habits of the player.

3. Thinking and judgement: this is central to the AI animation game system. It represents all the action benchmarks of the animation game characters and guides the animation game characters in terms of the judgements they should make when they encounter a certain situation. Strong AI animation games can become more humane and unpredictable, based on the information obtained.

4. Action: after passing the feeling and thinking judgement, the system will analyse a relative response to the current situation and instruct the animation game character to take action. Usually in animation games, one or more actions are set for animation game characters in advance, such as attack, escape, rest, patrol, gathering, etc., and animation game characters are analysed and judged by the AI animation game system, according to the situation.

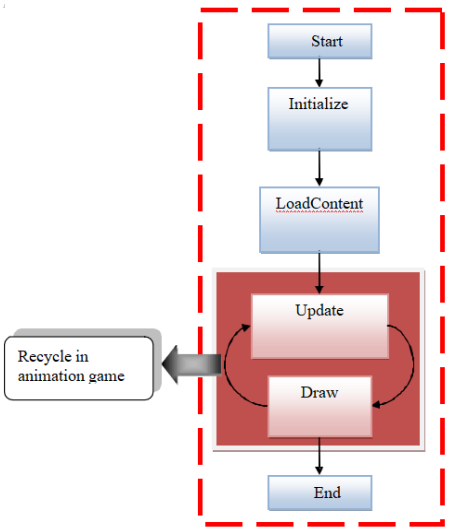

Fig. 2 Diagram of XNA animation game process architecture

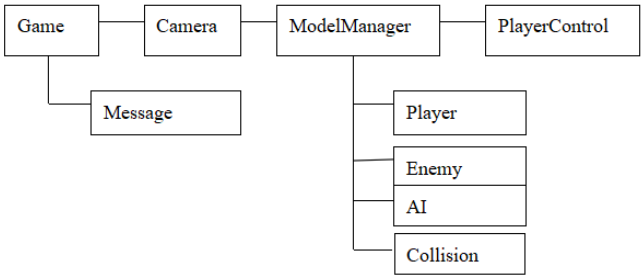

Fig. 3 Diagram of animation game architecture

In order to respond to various needs in animation games, there may be more than one AI system at the same time. These systems can operate separately or independently, or they can be related to one other.

3.5. AI Animation Game Information

The AI animation game system collects the information that is available or required in the animation game. Collecting information forms the basis and the start of the AI system. Without sufficient information, the AI system cannot create more effective and changeable thinking and judgements. Too much junk information will increase the calculation of the AI system and affect the performance of animation games, therefore, the setting and control of animation game information is very important. This information should be provided to the AI system and the type of information must be planned when the system is constructed. The animation game information used in the AI animation game system is divided into four categories.

1. World information: this type of information represents the character's information in the animation game world, such as the character's position, direction and height in the animation game world.

2. Character information: this type of information represents basic information relating to the character, such as strength and blood volume. The basic information of the character is used as a calculation basis for other information. For example, power will affect the attack damage value of the character; the higher the character, the higher the damage value it can cause.

3. Action information: this type of information not only refers to the attacking moves of the character, but also refers to the various actions of the character in the animation game, such as dodging, moving and attacking. This type of information transfers the player's current action information to the AI character, so that the AI character can determine the corresponding action. For example, when the AI character learns that a player is making an attack, the AI character may use a dodge action, when the player is on the move or on standby.

4. Situation information: this kind of information is usually used to indicate a certain situation relating to the AI character or to the player. When a player is attacked, there will be a period of time when the player cannot be attacked again, therefore, this

function must provide the collision detection system with information as to whether a character has collided with something.

4. AI System Implementation

4.1. 4.1 Main Program of Animation Game

The AI system is the main source, therefore, the main program of the animation game is built to provide the AI system with operation and testing functions. As shown in Fig. 4, the main program of animation game is divided into five parts according to its functions, and each part contains one or more categories.

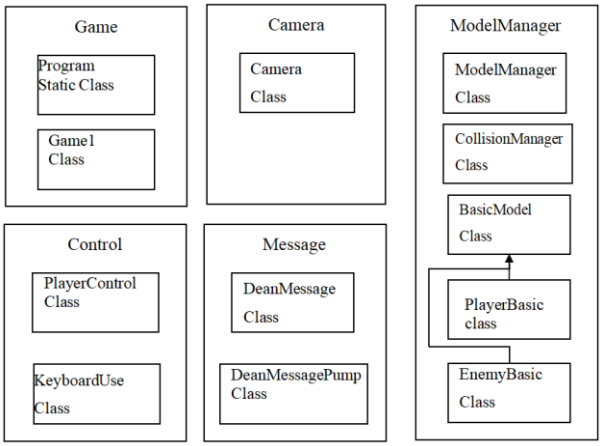

Fig. 4 Five major parts of the main program of the animation game

Object-oriented Programming (OOP) is an important concept of modular programming. An object is a combination of data and functions and the advantage of the modular object is that it can be repeated. OOP comprises three concepts, encapsulation, inheritance and polymorphism, which are explained as follows:

1. Encapsulation: to combine data, data processing procedures and functions into objects. The definition of an object in C# language is a class, which belongs to an abstract data type. To define an object is to define a new data type for the programming language.

2. Inheritance: this is the reuse of objects. When a new category is defined, it can inherit the data and methods of other categories; the new category can also add or replace the data and methods of inherited objects.

3. Polymorphic: if a category needs to process a variety of different data types, there is no need to create exclusive categories for different data types. This can directly inherit

the basic category to create a method of the same name, so as to process different data types.

(1) **Game**: this is the skeleton part of the animation game; its main function is to control the animation game process and set the animation game environment. Game contains two categories, Program and Game1, which are automatically generated by the system when a new XNA animation game project is created. The Program and Game1 categories will be described separately below:

1. Program: the function of the Program is quite simple, as it only contains a main function. This function is the first to be executed when the animation game starts to run. The main function declares an object of the Game1 category and executes the Run function of this object.

2. Game1: Game1 was originally the main loop function of the animation game; the main function is the initial setting and process control of the animation game. However, since the function of the template program has been used to allow other animation game components to operate independently, only the initial animation is left in Game1. The game environment has the function of building other animation game elements.

(2) **Camera**: the main function of the camera is to set and manage animation game cameras. It only contains a camera category, which is an animation game component, so it will update its status with the animation game time. Drawing in a 3D animation game world is like using a camera to record a video. The drawing of objects on the screen needs to be imaged through the parameters provided by the camera. The key camera parameters that affect imaging are the two matrix parameters, namely the view matrix and the projection matrix. The view matrix determines the position, facing direction and orientation of the camera in the animation game world. The projection matrix is used to determine the angle of view and the distance of the camera's visibility. Generally, the projection matrix does not need to be changed to its default value, unless the purpose is to change the visibility of the camera. The view matrix that needs to be constantly changed in animation games is determined by the three parameters of the camera position, the camera's gaze target point and the camera's upward vector. By substituting the three parameters into XNA, the CreateLook method in the function library can obtain the view matrix of the camera. If the direction of the camera is unknown in relation to the target, the player should add the camera position and the camera's facing direction to ensure that the camera is facing the target point.

(3) **ModelManager-animation game model manager**: 3D model objects are the foundation of 3D animation games. For example, animation game scenes and animation game characters, etc. are all created by 3D model objects. The function of the animation game ModelManager is to centrally manage all animation game model objects. This contains three main categories of objects, BasicModel (PlayerBasic and EnemyBasic are inherited from BasicModel), ModelManager and CollisionManager. The three categories are introduced below. BasicModel represents the basic model object category in animation games and other model objects will inherit from it. The purpose of establishing BasicModel is to unify the parts shared by all model objects in the animation games. This approach can avoid the issue of redefining the model objects each time, as shown in Fig. 5. When creating a new model object, allow it to inherit the BasicModel, then, the new model object can have the basic data and functions of the BasicModel and expand them as required. For example, PlayerBasic is the extension of

the model object representing the player and EnemyBasic is the extension of the model object representing the enemy.

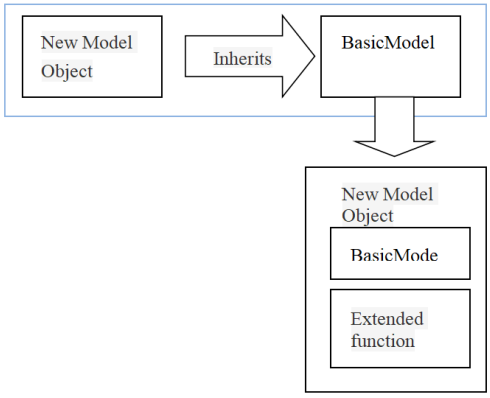

Fig. 5 A new model object inherited from BasicModel

4.2. AI system Program Description

The AI system of this study uses three AI technologies: FSM, FuSM and neural network-like technology. Since the FSM and the FuSM are quite similar in terms of basic architecture, these are two technologies which are built in the same module at the time.

(1) State machine module: this kind of FSM has two advantages. The first is high scalability. Nowadays, the scale of animation games is getting larger and the subsystems included in animation games are increasing. The main body of the AI systems of the animation game may be developed by two groups of people at the same time. In this case, the use of information becomes the communication method between the subsystems, making the subsystems independent, therefore, there is no need to worry about the information connection between the subsystems. The second point is to improve the efficiency of the system, as most AI characters are reactive. Usually, there is a change in the animation game that affects the behaviour of the AI character. Even if the AI character's behaviour does not change, the program will continue to run all the judgement programs, which will reduce efficiency. If the message appears, it is necessary to send the relevant message to the AI system when the animation game changes. The message-driven FSM uses messages as the driving force for transitions between states. In the AI system architecture, the AI manager is responsible for the decision-making element, while the state machine manager actually controls the state transition actions and state operations.

(2) Neural network module: this type of neural network can be divided into the feedforward network and the recurrent network. The characteristic of this type of network is that each layer can only pass forward and the difference between the recursive network and the feedforward network is that the recursive network allows the

network layer to pass back. In the AI design of animation games, most people choose to use the feedforward network. The reason is that the feedforward network is easier to understand and test than the recursive network, and in terms of system performance, due to the feedforward network, the path does not resemble a recursive network that requires multiple circumventions in the network, therefore, the system performance of the feedforward network is also improved. Based on these reasons, this study chose to use a three-layer feedforward network, which is divided into three network layers as the name suggests, namely the input layer, the hidden layer and the output layer.

After choosing the type of neural network, the input data, activation function, output data and training method must be planned:

1. Input data: for the neural network of the AI animation game system, the amount of input data can be controlled to a minimum, as the relatively small volume of input data also means that the neural network has a higher execution rate; too much data are surplus to requirements. The input data may cause a gap between the output of the neural network and the prediction.

2. Activation function: the activation function converts the total input value of the neuron into the output value of the neuron. The activation function used in this study is the most commonly used logistic function.

3. Output data: the output value of the similar neural network is usually the decision value that represents the expected network prediction. Since the activation function used is a logistic function, the output value will be between 0 and 1, but the actual usable range is between 0.1 and 0.9, so a value around 0.1 can be regarded as inactive and a value around 0.9 is active. In this way, provided that the output value is judged as active, it is possible to determine the decision of the neural network.

4. Training method: the purpose of training the network is to identify the weights of the interconnections between neurons, so that the input can achieve the expected output. There are two types of training methods commonly used in animation games, which are supervised learning and unsupervised learning. Unsupervised learning is independent training and does not require design. The author manually edits the training set, but the required technology is more sophisticated and more extensive, considering the difficulty and time factors, so this research chooses the easier-to-implement back-propagation method in supervised learning. Training is an optimization process, and the optimization method used by the inverse transfer method is to use the error method to minimize the error. This process can be divided into the following five steps.

Step 1: establish a training set containing input data and expected output values.

Step 2: set the initial value of the weight of the neural network to a random small value.

Step 3: pass the input data of the training set into the network and allow the network to calculate the output value.

Step 4: compare the output value calculated by the network with the expected output value and calculate the error between the two.

Step 5: adjust the weight value to reduce the error and repeat the process until the error is within an acceptable range.

In the optimization process, after calculating the cumulative error of all the input data in the training set and the expected output value, a judgement is made as to whether this has reached the acceptable range, and if not, the process is repeated. The formula for calculating the error is represented by formula 2 and 3, which are used by the output

neuron and the hidden layer neuron respectively; the input layer has no so-called error because the value of the neuron is given by:

$$\delta_i^{\ o} = (n_{di}^{\ o} - n_{ci}^{\ o})\, n_{ci}^{\ o}(1 - n_{ci}^{\ o}) \quad (2)$$

$$\delta_i^{\ h} = (\Sigma_{\delta j}^{\ o}\, w_{ij})\, n^{h}(1 - n^{h}) \quad (3)$$

where δ^o is the error of i-th output neuron in the output layer. $n_{di}^{\ o}$ is the expected output value of the i-th neuron in the output layer. $n_{ci}^{\ o}$ is the calculated output value of the i-th neuron in the output layer. δ h is the error of the i-th output neuron in the hidden layer. δ o is the error of the j-th output layer neuron, connected by the i-th hidden layer neuron. w_{ij} is the connection weight between the i-th hidden layer neuron and the j-th output layer neuron. n^{h} is the calculated output value of the i-th neuron in the hidden layer.

5. Implementation

The examples of AI system implementation are used to demonstrate the effectiveness of modular AI systems.

5.1. Example 1

Example 1 will use modular animation game objects to build a working AI character example. The following implementation process from scratch is explained in a series of steps and the implementation process is divided into three parts: pre-work, setting the main body of the animation game and establishing the AI system. The pre-work involves the process of creating a new animation game project and placing modular objects. The main body of the animation game is established to connect various functional objects of the main body of the animation game and to establish the animation game flow. Finally, the establishment of the AI system builds the corresponding AI function objects according to the set AI character reactions and actions, and connects them with the main body of the animation game.

(1). Pre-work: i) the first step is to create a new XNA animation game project. The newly created animation game project only contains Game1 and Program categories, and currently, the Game1 category only has a basic animation game empty structure without any content.

ii) the second step is to place the module object: after creating the animation game project, the main body of the animation game and the AI system module objects that have been made previously, are placed into the project.

(2). Setting the main body of the animation game: at present, the objects are still in separate conditions, therefore, it is necessary to start writing programs to connect the main parts of the animation game and to create players and AI characters. At this time, the AI characters are not connected to the AI system. The following describes the key programs in each object.

i) Game1: the most important function of Game1 in an animation game relates to the initial components. The animation game components that need to be created are Camera, ModelManager and PlayerControl. Therefore, these three objects should be created at

the initialize function stage and added to components, so as to connect them. The main loop of the animation game begins to operate. After the animation game components are established, the associations between the components must be connected. In the SceneInitialize function, the player character should be the target of the Camera, then the player character and the AI character in the ModelManager should be linked to the PlayerControl.

ii) ModelManager: creates model objects of players and AI characters in ModelManager, even though there is only one model object for both the player and the AI character. The module objects of the player and the AI character are created, then the state and drawing are updated with the ModelManager.

iii) PlayerControl: there are three kinds of player behaviours set in animation games, moving, attacking and dodging. The update function of PlayerControl will respond to the player's input every time it is updated. If this is true, the action will be executed; actions are restricted in order.

(3). Establishment of AI system: before establishing an AI system, the kind of AI roles to be established must be determined. The example below shows the creation of a close-range attacking AI character that includes fixed behaviours and variable behaviours. The AI roles linked to these behaviours are shown in Table 4.

Table 4. Example 1 Action behaviour of AI characters

AI character	Behaviour description
Patrol	Without encountering a player, the AI character performs a movement behaviour centred on the spawn point.
Chasing the player	When the player is within visual range, the AI character pursues the action of approaching the player.
Combat Movement	An AI character moves around a player-centred point while within combat distance.
Attack	When within combat distance with the player, the AI character carries on the attack action to the player.
Dodge	A fast lateral move performed by an AI character when he or she is under attack or needs to move quickly.

AI characters have five behaviour modes, which can be roughly divided into two parts: combat and non-combat. The combat element includes combat movement, attacking and dodging; the non-combat element patrols and chases the player. According to the aforementioned classification, the battle element is planned as a variable behaviour and the non-combat element as a fixed behaviour; the fixed behaviour uses an FSM and the variable behaviour uses a neural network. The plan is shown in Fig. 6.

EnemyAIManager: the AI manager connects the AI role and the AI system. Firstly, a category EnemyOneAIManager is created that inherits the EnemyAIClass, connects the EnemyOneAIManager in the EnemyOne category and updates the AIManager in EnemyOne's update function. As a result, EnemyOneAIManager is connected to the FSM. EnemyOneAIManager wishes to manage the condition information required by the state transition of the FSM and the input information required by the neural network. FSM: the connection between the AIMachine of the FSM and each state has been completed in EnemyOneAIManager. The following describes the two states of

EnemyChaseState and EnemyPatrolState; EnemyFightState uses neural networks internally.

EnemyPatrolState: this state controls the AI character's patrol action centred on the spawn point. After entering this state, the character will first calculate the distance between the current position and the centre of the circle, and will store it in the length parameter.

EnemyChaseState: this state controls the behaviour of the AI character chasing the player. The operation of this state is quite simple. Provided that the AI character is in this state, it will continue to move with the player as the target, until it leaves the state.

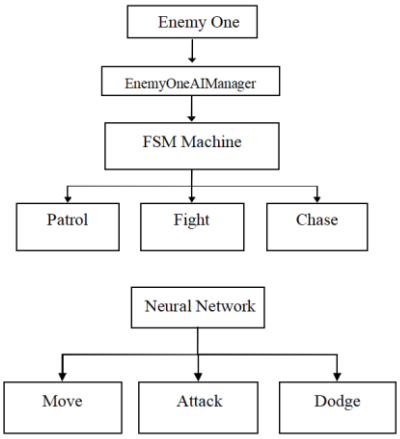

Fig. 6. Example 1 AI system planning

(4) The establishment of the AI system-neural network: the neural network in Example 1 is used to decide the combat behaviour of AI characters, so the neural network is placed in the EnemyFightState state of the AI character for use. There are three combat behaviours in the EnemyFightState state: move, dodge and attack. These three behaviours represent the output of a neural network, using input AI character emotions, current player actions and AI characters relative to the player. The three pieces of information relating to location use neural networks to determine the combat behaviour that should be executed, as shown in Fig. 7.

Before using the neural network, it is necessary to train in the neural network. The training method used here provides a supervised learning of the training set. The method needs to create a training set containing input and expected output data, as shown in Table 5.

The training process involves passing the input of the training set to the neural network to run, then comparing the obtained output with the expected output value in the training set to calculate the error. Then the inverse transfer method is used to update the weight of the input and the hidden layer; this process is repeated until all the data in the training set are calculated once, then, a judgement is made as to whether the accumulated error meets the expected value. If this is not the case, the training process is repeated until the error reaches the expected target. It takes time to train a neural network. If a large number of such training processes are repeated in an animation game,

this may cause delays in the animation game. Therefore, the neural network is built in the EnemyOne category to make the animation game in the first place. When it starts to read AI characters, the neural network must be trained first and the data stored externally, then the trained neural network data are read from the outside when the animation game is executed.

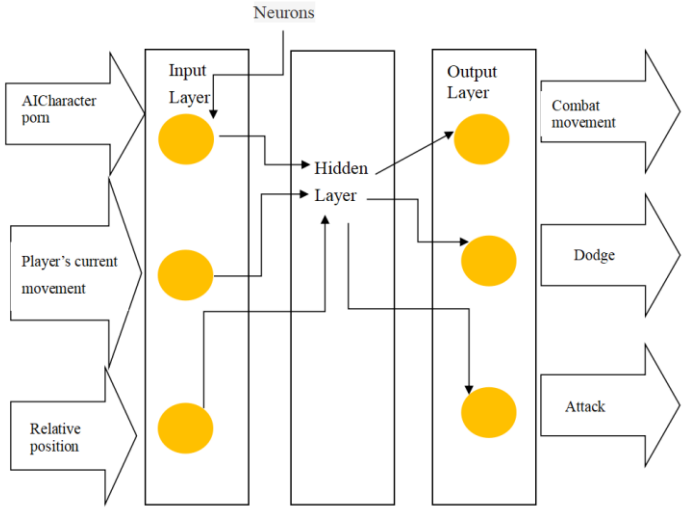

Fig. 7. The neural network in the implementation example 1

Table 5. Examples of training sets of neural networks

Input			Output		
AI character emotion	Player's current movement	Relative position	Combat movement	Dodge	Attack
0.0	0.1	0.1	0.1	0.9	0.1
0.1	0.2	0.1	0.9	0.1	0.1
0.3	0.1	0.4	0.1	0.9	0.1
0.5	0.1	0.2	0.1	0.1	0.9

Once the neural network has been built, the three combat behaviour functions, namely, move, attack and dodge are explained below.

1. Combat movement: the behaviour of move control is to allow the AI character to circle around the player. When the player approaches, the AI character moves back to keep a distance from the player.

2. Attack: the behaviour controlled by attack is an attack action. When the distance between the AI character and the player is greater than the attackable distance of the AI character, the AI character will move closer to the player and the attack action program will be executed when the distance is less than that.

3. Dodge: the behaviour controlled by dodge is the act of dodging. When the dodgeAnim parameter is false, the current position of the AI character should first be

saved to the prePosition parameter and dodgeAnim set to true, to avoid changing to prePosition. When the canDodge parameter of the AI character is true, this means that the AI character is currently able to perform a dodge action. The distance between the AI character's current position and the location to which it intends to move is judged, then the dodge action and the displacement will be performed; if the distance is considerable, the AI character may have already dodged when the distance was exceeded, therefore, the dodge action and movement are stopped at this time. The construction of Example 1 has been completed.

5.2. Example 2

Since many well conFig. parts can be used from Example 1, Example 2 will be simpler and faster to implement, and reuse is one of the benefits of modularity. The description of the settings that are will be omitted.

5.3. Set the Main Body of the Animation Game

The main function of the animation game is roughly the same as in Example 1. The difference is that in Example 2, a new spear class, inherited from BasicModel, is added. Spear is used to represent the spear thrown by the AI character, and the spear needs to be independently updated after being thrown. Therefore, a basic model series, relating to the spear, is set up in the ModelManager, and the spear is also added to the Update and Draw functions: operation and establishment of AI system-AI manager and FuSM. In the AI system during the implementation of Example 2, the aim was to highlight the modular reusability and design flexibility, therefore, during the design process, part of the architecture of Example 1 was combined with the new architecture of Example 2. The aim in the second example was to implement an AI role that was significantly different from the first example. Therefore, the second example selected the implementation of a long-range, attacking AI role. Table 7 below shows the behaviour list of the second AI role.

The AI character in Example 2 has four behaviour modes: patrol, chasing the player, combat movement and long-range attack. The two behaviours of patrol and chasing the player are the examples shown in Example 1, on the one hand, to demonstrate the reusability and on the other hand, because the two behaviours of patrolling and chasing players are almost necessary behaviours for AI characters in action animation games. Similar to Example 1, the four behaviours are divided into the combat element and the non-combat element. The non-combat element also uses FSM architecture to establish the two behaviours of patrol and chasing the player, but the difference is the use of the combat element. The FuSM is used to establish the two behaviours of combat movement and long-range attack. Fig. 8 is the AI system planning of Example 2.

1. EnemyAIManager: the EnemyTwoAIManager is created in the EnemyTwo category, as the AI manager of Example 2. The part of the EnemyTwoAIManager setting that connects the limited state machine is the same as in Example 1. The two states of EnemyPatrolState and EnemyChaseState are the objects created directly using Example

1. Only the combat state part is an additional EnemyFightTwoState state. The information processed by EnemyTwoAIManager is different from that in Example 1. The information here is used to determine whether the fuzzy state is enabled. The two states in the FuSM are based on the distance between the AI character and the player, so as to judge whether or not to enable, therefore, this parameter must be managed in EnemyTwoAIManager, in order to make the FuSM available for use.

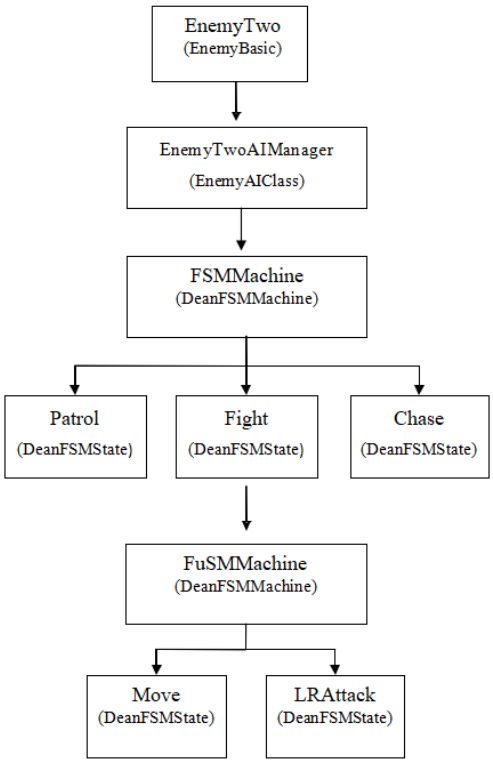

Fig. 8. Example 2 AI system planning

The following describes the two states of EnemyFuMoveState and EnemyFuAttackState, respectively.

1. EnemyFuMoveState: EnemyFuMoveState is the state of controlling the movement of combat. There are three behaviours. The first is to stand in one place, the second is to go around the centre of the circle with the player and the third is to move backwards when the player approaches, in an attempt to keep a distance from the player.

2. EnemyFuAttackState: the function of the FuSM is to allow the two states of EnemyFuMoveState and EnemyFuAttackState to operate independently, allowing the AI character to attack the player at a distance while moving. The behaviour of the EnemyFuAttackState is very simple and involves throwing a spear at the player at a fixed time.

Combining Two Implementation Examples

Example 2 shortens the implementation time considerably due to the modularity and quickly adds a new AI role by expanding Example 1. Then, the two examples are combined. Example 2 used many objects of Example 1 in its implementation and adds a small part of the program while maintaining the object structure, so Example 2 is an extension of Example 1. Therefore, we take Example 2 as the main body, and integrate a part of Example 1 in Example 2. Firstly, it is necessary to identify the objects that are different in Example 1 and Example 2, then the objects of both are gathered together. Following a comparison, the objects in Example 1, that are different from Example 2, are EnemyOne, EnemyOneAIManager and EnemyFightState. These three objects are added to the project in Example 2. The next step is to start to connect the program parts of the two examples. Firstly, the combination of the two examples are examined in the form of a framework diagram. As shown in Fig. 9, there is no conflict between the two examples and the only connecting part is EnemyOne. There is a link between EnemyTwo and ModelManager, therefore the only change required is to add a new AI role of EnemyOne to the EnemyModels list in ModelManager and the implementation of Example 2 has been completed.

Performance Evaluation and Discussion

This study used the XNA Game Studio, which simplifies the development of animation game programs in order to implement the AI system of action animation games, combined with AI technologies such as FSMs, FuSMs and neural networks, so that AI technology can be used. The results of this paper, namely, the concept of modular objects, allows the AI system to exhibit features such as scalability, reusability and design flexibility. The implementation process records and experience of this research can provide useful information for related research reference for both the creator and the producer. The research conclusions and contributions of this paper are as follows:

1. Achievements: an AI animation game system with reusability, design flexibility and expandability has been built, using the concept of object-oriented modular objects. As a result of the use of FSMs, FuSMs and neural networks, AI technology applications are added to combine traditional rule-based systems and learning adaptation systems, so as to increase the learning ability for traditional AI roles.

2. Two examples of AI roles with different behaviour patterns are implemented: the AI role behaviour in the first example is based on close-range attacks. The AI system architecture uses a combination of FSMs and neural networks and uses traditional rule-based methods. In Example 1, the implementation process from scratch verifies the feasibility of the AI system. Example 2 is completed practically through the combination of various built objects and through the combination of FSMs and neural networks. AI performance also achieved the expected results.

3. The AI character behaviour of the second example is aimed to be a long-range attack type and the combination of an FSM and an FuSM is used in the architecture of the AI system. The FSM element is built according to the first example. Fuzzy logic is used to allow the AI character to change its movement speed and movement method as the player approaches. The behaviour of AI characters is designed to have more

variability, and Example 1 is combined with Example 2 by means of simple steps, so that the two AI characters can be presented in the animation game at the same time. During the implementation of Example 2, many objects created in Example 1 were reused. This verifies the reusability and expandability in this research. These two features allow animation game designers to continuously reuse previously built objects and can also expand the functions of the objects to create an animation game design.

4. When an animation game designer tries out different AI technologies, he/she simply needs to expand or modify the original objects, without the need for a complete redesign. The difference between the two examples in the AI system architecture aims to highlight the design flexibility achieved by this research study. Since each AI system can operate independently, the animation game designers can integrate different AI technologies according to their needs and all AI systems can operate within animation games at the same time without affecting one other. Therefore, this research has reached the expected goals in terms of the implementation results.

5. XNA simplifies the effectiveness of animation game programming: XNA can simplify animation game design during the process of implementing the AI system and can explain and record the implementation process, so as to provide a reference for related researchers and producers.

The previously created animation game module objects are used to gradually build a prototype animation game, and the FSM and neural network technology is used to create an AI character that combines fixed and variable behaviours, as demonstrated in Example 1. The feasibility of the AI system and the combined effect of modular objects are discussed. During the implementation of Example 2, another AI character was quickly created by following the system established in Example 1 and completed using a different combination of AI technologies from Example 1, which demonstrates the reusability of the AI system, as well as design flexibility. The combination of the two examples shows the independence of the AI system. Combining the above characteristics can allow the AI system of this study to quickly create diversified AI roles, enabling each AI role to have sufficient design flexibility, which is not limited to a fixed architecture.

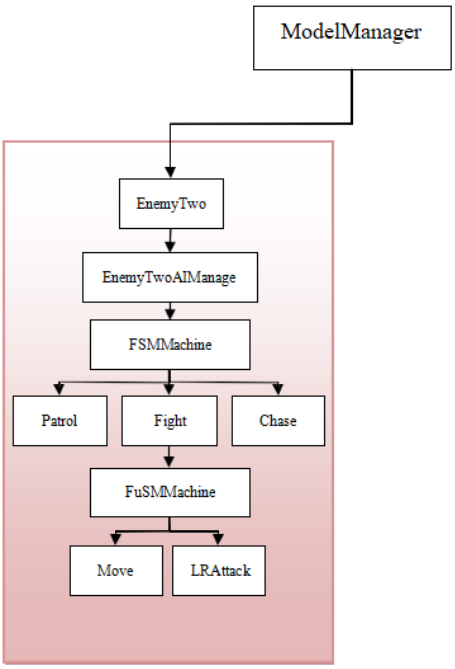

Fig. 9 Architecture diagram after the two examples are combined

6. Conclusion and Future work

This study used XNA Game Studio, which simplifies the development of animation game programs, so as to implement the AI system of action animation games, combined with AI technologies such as FSMs, FuSMs and neural networks, making use of AI technology. The results of this paper demonstrate the concept of modular objects, which allow the AI system to exhibit features such as scalability, reusability and design flexibility. Through the implementation process, records and experience of this research can provide useful information for related research, both for the creator and the producer. The research conclusions and contributions of this paper are as follows:

1. Achievements: an AI animation game system has been built with reusability, design flexibility and expandability, using the concept of object-oriented modular objects. By making use of FSMs, FuSMs and neural networks, AI technology applications are added to combine traditional rule-based systems and learning adaptation systems to increase the learning ability of traditional AI roles.

2. Two examples of AI roles with different behaviour patterns are implemented: the AI role behaviour in the first example is based on close-range attacks and the AI system architecture uses a combination of FSMs and neural networks, as well as traditional rule-based methods. In Example 1, the implementation process from scratch verifies the feasibility of the AI system and Example 2 is a practical example, created from a

combination of various built objects and the combination of FSMs and neural networks. AI performance has also achieved the expected results.

3. The AI character behaviour of the Example 2 is planned to be a long-range attack type, and the combination of an FSM and an FuSM is used in the architecture of the AI system. The FSM element is built according to the first example. Fuzzy logic is used to allow the AI character to change its movement speed and movement method as the player approaches. The behaviour of AI characters is given more variability and Example 1 and Example 2 are combined through simple steps, so that the two AI characters can be presented in the animation game at the same time. During the implementation of Example 2, many objects created in Example 1 were reused, which verifies the reusability and expandability in this research. These two features allow animation game designers to continuously reuse previously built objects and also to expand the functions of the objects to create an animation game design.

4. When an animation game designer aims to try out different AI technologies, he/she only needs to expand or modify the original objects without the need for a complete redesign. The difference between the two examples in the AI system architecture is to verify the design flexibility that this research expects to show. Since each AI system can operate independently, animation game designers can integrate different AI technologies according to their needs, and all AI systems can operate within animation games at the same time, without affecting one other. Therefore, this research has reached the expected research goals in the implementation results.

5. XNA simplifies the effectiveness of animation game programming: XNA can simplify animation game design by implementing the AI system and explaining and recording the implementation process to provide a reference for related researchers and producers.

The following are limitations of the AI system in this research study. 1. The lack of information hiding: the purpose of information hiding is to prevent the important information inside the object from being arbitrarily changed, since changing the information may damage the internal operating structure of the object. 2. The lack of a scripted design: at present, the parameter settings of the AI system must be directly modified in the program. When the animation game is large in scale, this will be costly in terms of compiling the project time. Future work: future research work will focus on the visual design of the system, as the programmers only provide technical support. The actual AI animation game is usually designed by the planners. If the planners in general have no knowledge of programming, it is relatively important to build an AI system that they can use. Therefore, creating a visual editor for the AI system will be one of the focuses of future research.

References

1. Habibie, I., Holden, D., Schwarz, J., Yearsley, J., Komura, T.: A recurrent variational autoencoder for human motion synthesis, Proc. 28th Brit. Vis. Conf. (BMVC), (2017).
2. Harvey, F., Pal, C.: Semi-supervised learning with encoder-decoder recurrent neural networks: Experiments with motion capture sequences, Comput. Sci., 3, 553-562 (2015).
3. Holden, D., Habibie, I., Kusajima, I., Komura, T.: Fast neural style transfer for motion data, IEEE Comput. Graph. Appl., 37(4), 42-49 (2017).

4. Holden, D., Saito, J., Komura, T.: A deep learning framework for character motion synthesis and editing, ACM Trans. Graph., 35(4), 138. (2016).
5. Kwon, T., Shin, S.: Motion modeling for on-line locomotion synthesis, Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation, 29-38 (2005).
6. Makuch, E.: Tomb Raider 1-5 hit Steam". GameSpot. Archived from the original in 2019. Retrieved 14 (2020).
7. Mandery, C., Terlemez, O., Do, M., Vahrenkamp, N., Asfour, T.: Unifying representations and large-scale whole-body motion databases for studying human motion, IEEE Trans. Robot., 32(4), 796-809 (2016).
8. Martinez, J.: Black, M., Romero, J.: On human motion prediction using recurrent neural networks, Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2891-2900 (2017).
9. Mehta D, et al.: Monocular 3D human pose estimation in the wild using improved CNN supervision, Proc. Int. Conf. 3D Vis., pp. 506-516. (2017)
10. Merel J, et al.: Learning human behaviors from motion capture by adversarial imitation, Available: https://arxiv.org/abs/1707.02201. (2017)
11. Roberts, S.: Age of Empires 4 is 'making good progress', and Microsoft will talk about it later this year". PC Gamer. Archived from the original in June, 2019. Retrieved 12 June, (2019).
12. Sato: Final Fantasy XIV Reaches 14 Million Adventurers Worldwide. Siliconera. Retrieved (2018).
13. Ke X, Zou J, Niu Y. End-to-End Automatic Image Annotation Based on Deep CNN and Multi-Label Data Augmentation. IEEE Transactions on Multimedia, 21(8): 2093-2106. (2019)
14. Shin S, Kim C: Human-like motion generation and control for humanoid's dual arm object manipulation, IEEE Trans. Ind. Electron., vol. 62, no. 4, 2265-2276. (2015)
15. Skrebels J.: Super Mario 3D World + Bowser's Fury Announced for Nintendo Switch. IGN. Archived from the original in September, Retrieved 3 September. (2020)
16. Tan Q, Gao L, Lai Y, Yang J, Xia, S.: Mesh-based autoencoders for localized deformation component analysis, Proc. 30th AAAI Conf. Artif. Intell., 1-8. (2018)
17. Tom, M.: XNA Game Studio 4.0 Programming: Developing for Windows Phone 7 and Xbox 360. Pearson Education. ISBN 9780132620130. (2010)
18. Cheng, H., Wu, L., Li, R., et al.: Data recovery in wireless sensor networks based on attribute correlation and extremely randomized trees. Journal of Ambient Intelligence and Humanized Computing, 12(1): 245-259. (2021)
19. Cheng, Y., Jiang, H., Wang, F., et al.: Using High-Bandwidth Networks Efficiently for Fast Graph Computation. IEEE Transactions on Parallel and Distributed Systems, 30(5): 1170-1183. (2019)
20. Dai, Y., Wang, S., Chen, X., et al.: Generative adversarial networks based on Wasserstein distance for knowledge graph embeddings. Knowledge-Based Systems, 190: 105165. (2020)
21. Feng Y, et al.: Mining spatial-temporal patterns and structural sparsity for human motion data denoising, IEEE Trans. Cybern., vol. 45, no. 12, 2693-2706. (2015)
22. Fu, Y. G., Huang, H. Y., Guan. Y., et al.: EBRB cascade classifier for imbalanced data via rule weight updating. Knowledge-Based Systems, 223: 107010. (2021b)
23. Fu, Y. G., Ye, J. F., Yin, Z. F., et al.: Construction of EBRB classifier for imbalanced data based on Fuzzy C-Means clustering. Knowledge-Based Systems, 234: 107590. (2021a)
24. Fu, Y. G., Zhuang, J. H,, Chen. Y, P., et al.: A framework for optimizing extended belief rule base systems with improved Ball trees. Knowledge-Based Systems, 210: 106484. (2020)
25. Li Z, Zhou Y, Xiao S, He C, Huang Z, Li, H.: Auto-conditioned recurrent networks for extended complex human motion synthesis, Available: https://arxiv.org/abs/1707.05363. (2017)

26. Li, X. Y., Lin, W., Liu, X., et al.: Completely Independent Spanning Trees on BCCC Data Center Networks with an Application to Fault-Tolerant Routing. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 33(8): 1939-1952. (2022)
27. Liu, G., Chen, X., Zhou, R., et al.: Social learning discrete Particle Swarm Optimization based two-stage X-routing for IC design under Intelligent Edge Computing architecture. Applied Soft Computing. 10, 107215. (2021)
28. Liu, G., Chen, Z., Zhuan,g Z., et al. A unified algorithm based on HTS and self-adapting PSO for the construction of octagonal and rectilinear SMT. Soft Computing, 24(6): 3943-3961. (2020a)
29. Liu, G., Zhang, X., Guo, W., et al.: Timing-Aware Layer Assignment for Advanced Process Technologies Considering Via Pillars. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(6): 1957-1970. (2022a)
30. Liu, G., Zhu, Y., Xu, S., et al.: PSO-Based Power-Driven X-Routing Algorithm in Semiconductor Design for Predictive Intelligence of IoT Applications. Applied Soft Computing, 114: 108114. (2022b)
31. Liu, N., Pan, J. Sun, C., et al.: An efficient surrogate-assisted quasi-affine transformation evolutionary algorithm for expensive optimization problems. Knowledge-Based Systems, 209: 106418. (2020c)
32. Lu, Z., Liu, G., and Wang, S.: Sparse neighbor constrained co-clustering via category consistency learning. Knowledge-Based Systems, 201, 105987. (2020)
33. Wang, S., Wang, Z., Lim, K. L., et al.: Seeded random walk for multi-view semi-supervised classification. Knowledge-Based Systems, 222: 107016. (2021)
34. Xia, S., Wang, C., Chai, J., Hodgins, J.: Realtime style transfer for unlabeled heterogeneous human motion, ACM Trans. Graph., vol. 34, 119.. (2015)
35. Yu, Z, Zheng, X, Huang, F, et al.: A framework based on sparse representation model for time series prediction in smart city[J]. Frontiers of Computer Science, 15(1): 1-13. (2021)
36. Yumer, M., Mitra, N.: Spectral style transfer for human motion between independent actions, ACM Trans. Graph., vol. 35, no. 4, 137. (2016)
37. Zhang, H., Li, J. L,, Liu, X. M., et al.: Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection[J]. Future Generation Computer Systems, 122: 130-143. (2021a)
38. Zhang, Y., Lu, Z., and Wang, S.: Unsupervised feature selection via transformed auto-encoder[J]. Knowledge-Based Systems, 215: 106748. (2021b)
39. Zheng, X., Rong, C, et al.: Foreword to the special issue of green cloud computing: Methodology and practice[J]. Concurrency and Computation: Practice and Experience. 31(23): e5425. (2019)
40. Zhou, X., Zhu, M., Pavlakos, G., Leonardos, S., Derpanis, K., Daniilidis, K.: MonoCap: Monocular human motion capture using a CNN coupled with a geometric prior, IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 4, 901-914. (2019)
41. Zou, W., Guo, L., Huang, P., et al.: Linear time algorithm for computing min-max movement of sink-based mobile sensors for line barrier coverage. Concurrency and Computation: Practice and Experience, 34(2): e6175. (2022)
42. Zhong, S, Jia C, Chen K, et al.: A novel steganalysis method with deep learning for different texture complexity images. Multimedia Tools and Applications, 78(7): 8017-8039. (2019)
43. Pu L, Zhu D, Jiang H.: A 1.375-approximation algorithm for unsigned translocation sorting. Journal of Computer and System Sciences, 113: 163-178. (2020)
44. Shen, S., Yang, Y., Liu, X.: Toward data privacy preservation with ciphertext update and key rotation for IoT. Concurrency and Computation: Practice and Experience, e6729. (2021)
45. Pan, W., Zhao, Z., Huang, W., Zhang, Z., Fu, L., Pan, Z., Yu, J., Wu, F.: Video moment

retrieval with noisy labels. IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2022.3212900. (2022)
46. Ma, L., Zheng, Y., Zhang, Z., Yao, Y., Fan, X. Ye, Q.: Motion Stimulation for Compositional Action Recognition, IEEE Transactions on Circuits Systems and Video Technology, 2022, Early Access. (2022)
47. Fu, L., Zhang, D. and Ye, Q.: Recurrent Thrifty Attention Network for Remote Sensing Scene Recognition, IEEE Transactions on Geoscience and Remote Sensing, vol.59, no.10, 8257-8268. (2021)
48. Ye, Q., Huang, P., Zhang, Z., et al.,: Multi-view Learning with Robust Double-sided Twin SVM with Applications to Image Recognition, IEEE Transactions on Cybernetics, vol.52, no.12, 12745 - 12758. (2022)
49. Fu, L., Li, Z., Ye, Q., et al.: Learning Robust Discriminant Subspace Based on Joint L2,p- and L2,s-Norm Distance Metrics, IEEE Transactions on Neural Networks and Learning Systems, vol.33, no.1,130 -144. (2022)
50. Chen, X., Li, M., Zhong, H., Ma, Y. and Hsu, C.: DNNOff: Offloading DNN-based Intelligent IoT Applications in Mobile Edge Computing. IEEE Transactions on Industrial Informatics, 18(4): 2820-2829. (2022a)
51. Chen, X., Zhang, J., Lin, B., Chen, Z.: Katinka Wolter, Geyong Min. Energy-Efficient Offloading for DNN-based Smart IoT Systems in Cloud-Edge Environments. IEEE Transactions on Parallel and Distributed Systems, 33(3): 683-697. (2022b)
52. Chen, X., Hu, J., Chen, Z. Lin, B., Xiong, N., Min, G..: A Reinforcement Learning Empowered Feedback Control System for Industrial Internet of Things. IEEE Transactions on Industrial Informatics, 18(4): 2724-2733. (2022c)
53. Chen, X., Yang, L., Chen, Z., Min, G., Zheng, X. Rong, C.: Resource Allocation with Workload-Time Windows for Cloud-based Software Services: A Deep Reinforcement Learning Approach. IEEE Transactions on Cloud Computing, Publish Online, DOI: 10.1109/TCC.2022.3169157. (2022d)
54. Huang, G., Luo, C., Wu, K., Ma, Y., Zhang, Y, and Liu, X.: Software-Defined Infrastructure for Decentralized Data Lifecycle Governance: Principled Design and Open Challenges. IEEE International Conference on Distributed Computing Systems. (2019)

Rong Zhang received a bachelor's degree from Wuhan Institute of Technology, China, in 2006, and MS degree from Hubei University of Technology, Wuhan, China, in 2011. She is currently a lecturer with the School of Artificial Intelligence, Dongguan Polytechnic. Her major research interests include Artificial Intelligence, Internet of Things and animation game.

Received: July 19, 2022; Accepted: November 11, 2022.